MASTER EN SCIENCES INFORMATIQUES

Elaboration d'une plate-forme de corégistration multimodalités orientée objet

De Vreese, David

Award date:
2004

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique

Année académique 2003-2004

Élaboration d'une plate-forme de corégistration multimodalités orientée objet

David De Vreese

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

Résumé

Ce mémoire présente une première implémentation utilisable du projet de réalisation d'une plate-forme de corégistration multimodalités. Ce projet unique d'aide à la corégistration court depuis maintenant trois ans et les études théoriques menées par les étudiants précédents ont conduit, aujourd'hui, à cette première réalisation concrète.

L'ensemble des concepts liés à la corégistration seront tout d'abord présentés, afin de pouvoir ensuite formuler le problème inhérent à cette technique, présenter l'existant de la solution et enfin, présenter notre réalisation.

Cette réalisation se décompose en deux grandes parties. La première concerne la corégistration en elle même, et montre comment les algorithmes automatiques d'alignement ont pu être intégrés et utilisés efficacement dans la plate-forme. La seconde partie recouvre tout le problème de visualisation des images médicales et, en particulier, des images corégistrées.

Mots clés:

Corégistration, alignement, images numériques, visualisation

Abstract

This thesis presents a first usable implementation of the project concerning the realisation of a multimodalities coregistration platform. This unique project, that attempts to help for coregistration is now running for three years and the theoretical studies made by the former students have leaded, today, to this first concrete realization.

The whole of the concepts related to the coregistration will first of all be presented in order to be able to formulate the problem inherant with this technique, to present the existant of the solution and finally, to present our realization.

This realization breaks up into two large parts. The first one concerns the coregistration itself, and shows how the automatic alignment algorithms could be integrated and used effectively in the platform. The second part covers all the problem of visualization of the medical images and, in particular, the coregistered images.

Keywords:

Coregistration, alignement, digital images, visualisation

Remerciements

L'exercice du remerciement est toujours périlleux, mais je vais tâcher d'être le plus complet possible et de n'oublier personne.

Je tiens tout d'abord à remercier Monsieur Jean-Pol Leclercq et Hubert Meurisse, promoteur et co-promoteur pour ce mémoire, pour leur disponibilité, leur aide, les précieux conseils qu'ils ont pu me prodiguer pendant le stage et la rédaction de ce mémoire et, surtout, leurs encouragements et leur soutien.

Je n'oublierai évidemment pas tous mes amis, avec qui je suis arrivé au terme de ces trois années de maîtrise et avec qui j'ai passé les meilleurs moments de ma vie d'étudiant.

Merci à Dominique et Christelle pour leurs corrections orthographiques ☺.

Un merci tout spécial à Guillaume, sans qui la vie de la petite pièce aurait été beaucoup plus morne et difficile ☺.

Merci à mes parents et à toute ma famille pour leur soutien tout au long de mes études et, spécialement, pendant la rédaction de ce mémoire.

Merci à toi, Anne-Sophie, pour l'attention et l'amour que tu m'apporte tous les jours, qui m'aident à surmonter les moments difficiles.

Table des matières

Introduction

Description du problème:

La corégistration est une technique relativement nouvelle et, de par ce fait, les travaux réalisés sur le sujet sont encore peu nombreux. Les techniques de corégistration actuelles commencent à fournir des résultats plus qu'acceptables, mais elle ne sont, pour la plupart, pas encore utilisées voir même utilisables, du fait du haut degré de technicité nécessaire à leur mise en oeuvre. Il existe une multitude d'algorithmes disponibles permettant de corégistrer des images. Certains sont mieux adaptés à certaines tâches particulières et il est nécessaire de pouvoir **choisir le meilleur** à appliquer suivant le cas traité. De plus, leur utilisation n'est pas toujours des plus aisées. Il faut, en effet, pouvoir leur **fournir les paramètres nécessaires** afin de pouvoir optimiser le recalage. Un autre point important à souligner est qu'ils ne travaillent pas tous avec les **même formats de fichiers** ou avec les **mêmes conventions de représentation des images** utilisées dans l'hôpital. Toutes ces raisons, et bien d'autres encore, ont fait naître un projet au sein de l'hôpital Universitaire de Mont-Godinne. Ce projet a pour but de mettre en place une plate-forme de corégistration multimodalités, autrement dit, un programme muni d'une interface agréable et facile à utiliser, permettant de guider les médecins dans la tâche de corégistration ou, mieux encore, de l'effectuer d'une façon tout à fait automatique, indépendamment des détails techniques, afin de pouvoir, d'une manière qui leur serait la plus transparente possible, corégistrer les images choisies. Le projet ne s'arrête pas là: outre les algorithmes de corégistration automatiques, la plate-forme pourra disposer de ses propres outils manuels, permettant aux médecins d'effectuer une pré-corégistration, afin que l'algorithme automatique ait plus de chance de mener sa tâche à bien. Enfin, tout un système de visualisation des images médicales et, plus particulièrement, des images coregistrées pourra être mis en place, pour présenter les résultats obtenus.

Plan du mémoire:

Dans **la première partie**, nous poserons les concepts de base nécessaires à la compréhension de ce qu'est la corégistration. Nous présenterons, dans un premier chapitre, la notion d'images numériques et, plus précisément, celle d'images médicales. Dans le chapitre suivant, nous découvrirons la provenance de telles images, en présentant les différentes modalités que l'on retrouve en imagerie médicale. Un dernier chapitre viendra conclure cette partie introductive en définissant formellement la corégistration et en présentant ses mécanismes de base.

La seconde partie sera consacrée à la description du projet. Son utilité et son état d'avancement y seront présentés, nous donnant les bases pour notre propre réalisation.

La troisième partie présentera les différents outils et technologies utilisés lors de l'implémentation de la plate-forme. Nous y découvrirons les avantages et inconvénients de ceux-ci et, les raisons qui nous ont poussé à les utiliser ou à les laisser de côté.

La quatrième partie décrira l'application réalisée, autrement dit, l'apport qui a été fait aux travaux des années précédentes.

Enfin, **la cinquième partie** sera consacrée aux discussions et critiques que l'on pourra porter sur la réalisation.

1 Concepts de base

1.1 Les images

La définition de ce qu'est une *image numérique*, et plus précisément, de ce qu'est une *image médicale* est quelque chose d'indispensable à l'appréhension du terme *corégistration.* Cette partie restera néanmoins assez brève, pour ne pas embrouiller le lecteur et pouvoir rentrer au plus vite dans le vif du sujet.

1.1.1 Une image numérique

Le terme "image numérique", dans sa définition la plus vaste, désigne l'ensemble des images acquises, créées, traitées et stockées sous forme binaire. Elle proviennent d'une multitude d'appareils tels que l'appareil photo numérique, le scanner, les cartes d'acquisition,...ou sont créées directement à l'aide de programmes informatiques spécialisés. La forme la plus commune d'images numériques rencontrée est l'image de type *BitMap*. Ce type d'image est constitué d'une matrice (2D ou 3D) de points appelés *pixels*. [W-IntNum]. Le terme *pixel* est la forme abrégée de "picture element". Il représente la plus petite surface homogène constitutive d'une image enregistrée. Chaque pixel est donc d'une couleur définie et trouve sa place dans l'image grâce à la matrice associée à celle-ci.

Figure 1 : Agrandissement d'une partie de l'image pour mettre en évidence les pixels constituant celle-ci

Ce type d'image est le plus utilisé dans les applications de tous les jours (web, photo, ...)

Le deuxième type d'images numériques est l'image vectorielle. Ce type d'image est différent des images bitmap par le fait qu'il n'est plus codé à l'aide d'une matrice gérant la place des pixels au sein de l'image. L'image vectorielle est constituée d'un ensemble de formes calculées par des formules mathématiques. Comme exemple simple, on peut citer le dessin d'un cercle qui, dans une image matricielle, ne sera pas décrit à l'aide d'une matrice et des pixels associés, mais à l'aide de formules mathématiques définissant sa taille, son emplacement, sa forme,...[W-ImgVect]

Bien qu'intéressant en soi, le sujet des images vectorielles ne sera pas approfondi car ce type d'image n'est pas celui qui nous concerne dans ce travail.

1.1.2 Une image numérique médicale

Pourquoi faire une distinction entre les images numériques "normales", utilisées tous les jours dans les applications diverses ou sur Internet, et les images numériques médicales? Si l'image numérique et l'image médicale se retrouvent dans la même section de ce travail, c'est que l'on peut certainement trouver un certain nombre de similitudes entre les deux. Et en effet, on peut dire que l'image médicale est en partie constituée d'une image numérique. Mais il n'y a pas que ça. Dans un service d'imagerie telle que la radiologie ou le scanner par exemple, plusieurs centaines, voire plusieurs milliers d'images, sont produites chaque jour. La gestion serait quasi impossible avec un format d'image commun tel que jpeg par exemple. On pourrait facilement les mélanger et les confondre, ce qui, dans de pareils cas d'utilisation, pourrait avoir des conséquences plus que désastreuses. C'est pour cela qu'en plus des données brutes nécessaires au codage de l'image, on lui a adjoint diverses informations textuelles sur la provenance de l'image (la modalité utilisée), des informations pratiques sur l'examen réalisé (l'image provient d'un pet-scan ou d'une IRM) et, sans doute le plus important, des informations sur le patient ayant subi l'examen. [M-Vae03]

Nous avons, jusqu'à maintenant, parlé des images médicales en deux dimensions. Une chose importante à souligner est qu'en imagerie médicale, il existe des dimensions supplémentaires à ces images. Nous retrouverons ainsi des images en trois dimensions, dans l'espace donc, et des images à dimensions temporelles. La dimension temporelle permet de réaliser des piles d'images représentant une même partie du corps mais à des moments différents. Cela permet d'observer l'évolution d'une maladie.

Dans une image en trois dimensions, la plus petite unité distinguable n'est plus le pixel mais le voxel. Il est identifié par des coordonnées en x, y et z le situant dans l'espace.

A l'instar des formats d'images numériques communs tels que jpeg, gif, bmp,... il existe une multitude de formats d'images médicales utilisés suivant leur provenance et l'utilisation que l'on en fait. A titre d'exemple, on peut citer le fichiers VoxBo, Analyse et, ceux qui nous intéressent le plus dans ce travail, les fichiers *DICOM.*

1.1.3 DICOM

DICOM (pour ***D**igital **I**maging and **Co**mmunications in **Me**dicine*) est le nom d'une norme utilisée pour enregistrer les images médicales sur support numérique. Ce format fut créé en 1985 par la *National Electrical Manufacturers Associations* afin de permettre un échange aisé et simplifié des images entre les machines de constructeurs différents.

DICOM a ceci comme particularité qu'il ne représente pas seulement un format de fichier médical mais également un protocole de communication réseau permettant l'échange de ces derniers. [W-NecDic]

Un fichier DICOM est un fichier séquentiel. Il fonctionne avec un système de balises et a la structure suivante:

- L'identifiant de la machine ayant servi à faire l'examen
- Les informations sur le patient
- Les informations sur l'acquisition de l'information
- Les informations en rapport avec l'examen
- Les informations concernant l'image elle-même et la façon dont elle est codée
- Et enfin, les données brutes de l'image, c'est-à-dire la valeur des pixels.

Enfin, les données concernant les pixels peuvent être compressées par plusieurs méthodes (jpeg par exemple), mais ça sera rarement le cas. Les images médicales ont en effet besoin d'un maximum de précision afin de permettre un diagnostic précis. La compression pourra néanmoins se révéler utile lors de montage d'images en petites vidéos par exemple.

Ce qui précède est tout ce qui nous intéresse au sujet de ce format de fichier, le but de ce travail n'étant pas de réaliser un décodeur DICOM car celui-ci existe déjà et a donc été utilisé lors de la programmation de la plate-forme. Pour plus d'informations sur ce format, le lecteur pourra se référer à [W-NorDic].

Si, pour le moment, nous avons introduit les images médicales, il est peut-être encore un peu difficile pour un non-initié de pouvoir se donner une représentation précise de ce qu'elles sont réellement, d'où elle proviennent, de comment elle sont créées. C'est pourquoi, avant de pouvoir enfin introduire le terme *Corégistration*, il nous faut également parler des différentes *modalités* rencontrées en imagerie médicale. Ou du moins, des plus utilisées.

1.2 Les modalités en imagerie médicale

Comme l'avons laissé entendre en 1.1.2 , les images médicales proviennent de plusieurs modalités différentes. Dans ce chapitre, nous allons les présenter brièvement et montrer le type d'image qu'elles produisent.

1.2.1 La radiographie

En 1895, un physicien du nom de Wilhem Conrad Röntgen découvrait le principe de la radiologie et lançait ainsi un moyen révolutionnaire d'explorer le corps humain sans devoir y faire la moindre incision. Sa découverte sur les rayons X s'appuyait sur quatre observations majeures:

- Les rayons X sont absorbés par la matière; leur absorption est fonction de la masse atomique des atomes absorbants
- Les rayons X sont diffusés par la matière; c'est le rayonnement secondaire ou rayonnement de fluorescence
- Les rayons X impriment la plaque photographique
- Les rayons X déchargent les corps chargés électriquement

Il y a encore peu de temps, l'image issue de la radiographie était imprimée à partir d'un film argentique traversé par les rayons X ayant, au préalable, traversé les organes à étudier. [W-RadImg]. Avec l'avènement des technologies informatiques, la radiologie en général a suivi une évolution vers la numérisation des examens. Plus besoin de ces coûteux et encombrants films; dématérialisation et traitement post-examinatoire sont désormais de mise.

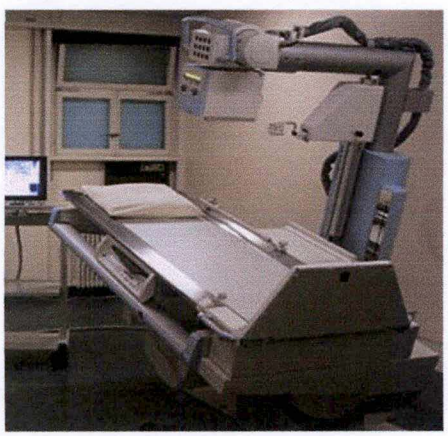


Figure 2 : Appareil radiographique

Les images produites par la radiographie sont de ce type:

Figure 3 : Cliché de crâne, de coté

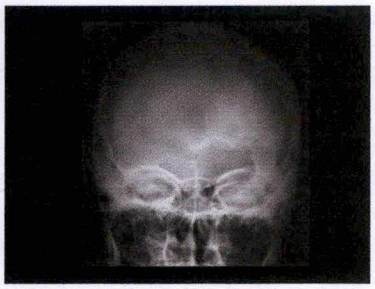


Figure 4 : Cliché de crâne, de face

1.2.2 L'IRM

Les initiales I.R.M signifient ***Imagerie*** par ***Résonance Magnétique***. Cette technique est relativement récente et a pour particularité de ne pas utiliser de rayons X mais les propriétés magnétiques du corps humain. De plus, cet appareil permet une étude *multiplanaire* permettant d'explorer la quasi-totalité du corps humain. L'examen reste assez coûteux et ne sera donc prescrit que pour certaines pathologies. Généralement, pour les explorations neurologiques et ostéo-articulaires.

Figure 5 : Appareil de résonance magnétique

Les images produites par de tels appareils sont de ce type:

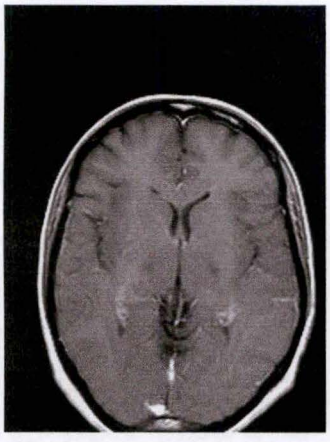


Figure 6 : Image axiale provenant d'une I.R.M

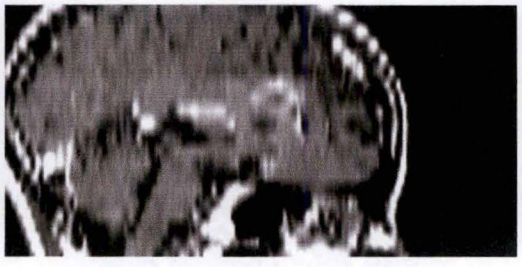


Figure 7 : Image sagittale provenant d'une I.R.M

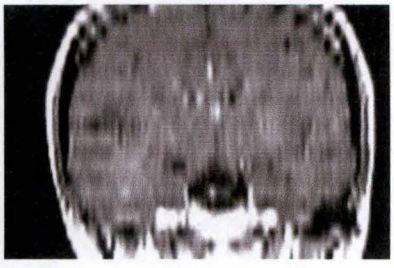


Figure 8 : Image coronale provenant d'une I.R.M

1.2.3 Le CT – Scan ou Computed tomography

Cette technique est semblable à la radiologie par le fait qu'elle utilise les rayons X. Ces rayons sont envoyés par un tube tournant autour du patient. Le film servant normalement aux radiographies étant ici remplacé par des capteurs chargés de récupérer les informations afin de pouvoir les traiter et reconstruire ainsi les images.

Figure 9 : Scanner

Le scanner produit ce que l'on appelle une *imagerie en coupe*, il fournit donc une pile d'image représentant les différents niveaux du corps du patient.

Voici le type d'images que l'on obtient avec le scanner:

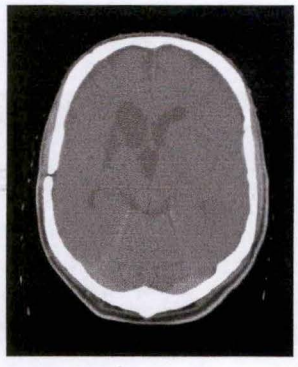


Figure 10 : Image axiale provenant d'un scanner

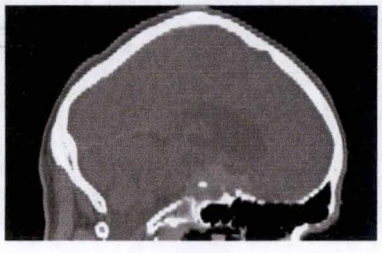


Figure 11 : Image sagittale provenant d'un scanner

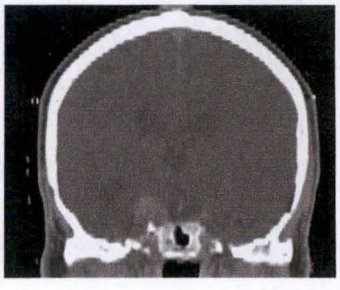


Figure 12 : Image coronale provenant d'un scanner

1.2.4 Le Pet-Scan

Le pet-scan pour ***Positron Emission Scanner*** est un appareil relativement récent dans le domaine du diagnostic clinique. Sa principale utilisation réside en la recherche de métastases et dans le suivi des patients après certains cancers.

L'imagerie Pet-Scan se base sur l'injection de ce que l'on appelle FDG (***Fluoro-Désoxy-Glucose***) au patient. Ce liquide, que l'on nomme *traceur*, va servir à repérer les zones ayant une activité métabolique intense. Les tumeurs sont en effet consommatrices de telles substances et vont absorber le liquide radioactif. Grâce à la machine, on va alors pouvoir observer le cheminement du liquide à l'aide de l'émission de positrons.

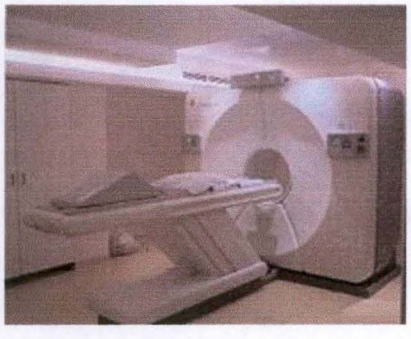


Figure 13 : Pet-Scan

Voici le type d'images que l'on peut obtenir avec une telle technique:

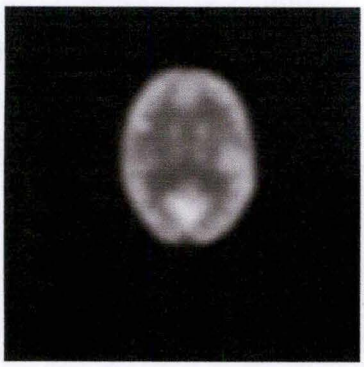


Figure 14 : Coupe axiale provenant d'un pet-scan

Figure 15 : Coupe sagittale provenant d'un pet-scan

Figure 16 : Coupe coronale provenant d'un pet-scan

Les techniques sus-citées, ne sont pas exhaustives mais montre bien l'arsenal mis à la disposition du médecin. Pour d'avantages d'informations sur le sujet, le lecteur est invité à consulter [W-IMED].

Les plus attentifs auront sans doute remarqué que l'on se trouve face à deux types d'images assez différents. Les images issues de la radiographie et de la résonance magnétique n'ont pas le même aspect que les images issues du scanner et du pet-scan. Les premières sont nommées *images anatomiques,* tandis que les secondes sont appelées *images fonctionnelles.*

Les images anatomiques permettent de mettre en évidence la structure anatomique du corps humain, de visualiser la structure des tissus organiques, comme si l'on prenait une photographie de l'organe. La taille, la position et la consistance des objets dans l'image sont d'une excellente précision.

Les images fonctionnelles, par contre, ne montrent pas les choses telles que l'œil serait apte à le faire mais sont créées à l'aide d'algorithmes mathématiques de construction d'images en fonction des données fournies par les différentes modalités. Au lieu de se concentrer sur l'aspect morphologique, les images fonctionnelles vont pouvoir rendre compte d'aspects physiologiques tels que l'activité métabolique des tissus, le parcours du sang, ... [M-Van98]

Devant ce panaché de techniques et de types d'images, on se rend bien compte que la tâche du médecin n'est pas des plus aisées. Il lui faut manipuler des images de taille, de

couleur et même d'aspect différent, afin de pouvoir poser son diagnostic. De plus, comme nous venons de le voir, les images n'ont pas toutes la même signification. Certaines se servent de l'intensité de couleur des pixels pour représenter les zones à forte activité métabolique tandis que d'autres associent cette intensité à la composition du tissu observé. C'est ici qu'intervient la *corégistration.* En réduisant au maximum les différences entre les images, elle va permettre d'établir plus facilement des relations entre elles et va donc donner au médecin des facilités pour la gestion de ces informations. La présentation de cette technique fera l'objet du chapitre suivant. Nous allons y présenter ses avantages et son fonctionnement plus en détail.

2 Etat de l'art

2.1 La corégistration

Maintenant que les bases techniques ont été présentées, nous pouvons définir ce qu'est la corégistration. D'abord en posant clairement le problème et en décrivant la nécessité d'une telle technique. Ensuite, en exposant les mécanismes de base sous-jacents à l'application de cette technique.

2.1.1 Pourquoi?

Comme nous l'avons découvert en 1.2, il est possible, pour le médecin, d'obtenir nombre d'images différentes d'une même partie du corps humain, et ceci, afin de pouvoir poser un diagnostic des plus précis. On trouve des avantages dans chaque type d'images généré. Ainsi, si l'on soupçonne un cancer du poumon, une radiographie est tout d'abord effectuée. Celle-ci permet de fournir des photos, de face et de côté, de l'intérieur du poumon. Ensuite, un scanner est demandé, mettant à disposition du praticien une série de photos du poumon en coupe horizontale de 1 à 6 mm d'épaisseur. Ceci afin d'aider à localiser l'anomalie. Enfin, un pet-scan vient compléter l'examen, en offrant une vue fonctionnelle de l'organe. L'ensemble de ces examens fournit au médecin d'importants atouts dans la pose de son diagnostic et dans la proposition du traitement adéquat.

Le problème, comme nous avons pu nous en rendre compte avec les images du point 1.2, est que l'on n'obtient pas d'images *homogènes* d'une modalité à l'autre. Par "homogène", nous entendons des images de même taille (dans les trois dimensions de l'espace), de même orientation (l'appareil peut scanner dans un sens différent ou le patient peut ne pas être dans une position identique lors des deux analyses), de même position et de même nature. (La couleur des pixels ont des significations différentes suivant le type d'examen prodigué).

Toutes ces différences rendent la tâche du médecin complexe. Il doit être capable de se faire une représentation mentale des zones de correspondance entre les différentes images. Voici, par exemple, ce que peut avoir un médecin comme type d'images. Ces images proviennent de modalités différentes et donnent des vues d'un même organe (le cerveau en l'occurrence).

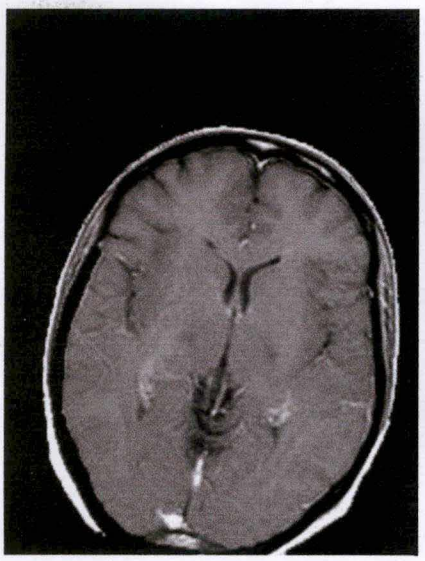


Figure 17 : IRM d'un cerveau

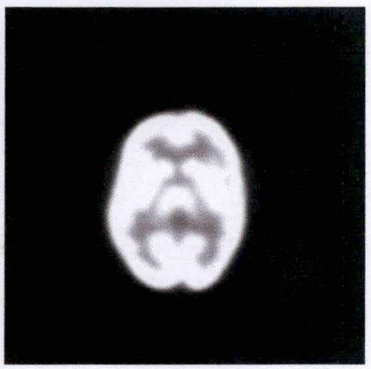


Figure 18 : Pet-scan d'un cerveau

Elles sont à bonne échelle. On constate la difficulté à se faire une idée précise des zones de correspondance.

C'est ici que la corégistration intervient. En appliquant des algorithmes mathématiques sur l'une des deux images (généralement la plus grande afin de ne pas devoir extrapoler des zones dans la plus petite pour la mettre à l'échelle), on aligne les images. On revient donc à la même orientation, à la même échelle, au même nombre de coupes. Cet alignement est illustré ci-dessous.

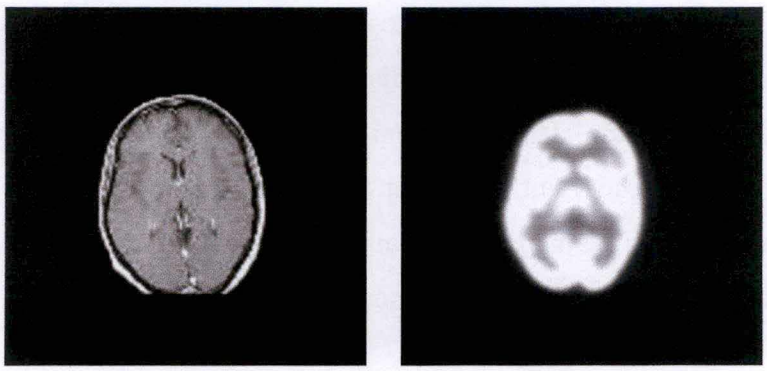


Figure 19 : Exemple d'images corégistrées

2.1.2 Comment?

La corégistration d'images n'est pas une technique figée et comprend nombre de méthodes différentes, certaines étant plus adaptés à des cas particuliers. Nous allons nous appuyer sur ce que propose [M-Sey02] pour comprendre la technique qui sera utilisée dans notre programme. Cette technique repose sur la séparation de la corégistration en les quatre parties citées page suivante.

1. Extraction de l'**espace des caractéristiques**, qui représente l'ensemble des informations extraites des images maîtres et esclaves afin de pouvoir les mettre en correspondance. On distingue généralement deux types de caractéristiques:

   - Les caractéristiques intrinsèques, qui sont des informations présentes dans l'image "naturellement" (une tache plus sombre, un trou,...). Ces caractéristiques sont, à leur tour, subdivisées en 2 grandes catégories:

     a. Les caractéristiques géométriques de l'image: les contours les plus marqués sont repérés et utilisés comme points de repère pour le réalignement.

     b. Les caractéristiques iconiques, qui se basent sur la mesure d'intensité des voxels, suivant la luminosité, le contraste, ...

   - Les caractéristiques extrinsèques, qui sont des repères artificiels apposés lors de l'examen pour des facilités de repérage.

   Ces caractéristiques servent donc de repère aux algorithmes afin de pouvoir effectuer les réalignements nécessaires.

2. Une fois les caractéristiques extraites, nous pouvons passer à la deuxième partie du travail qui consiste en la recherche d'une transformation spatiale qui permettra de mettre, le plus possible, les repères choisis sur les deux images en correspondance. Nous appellerons cette étape l'**espace de recherche** Plusieurs types de transformations existent, et il faut, suivant les cas, définir la transformation adéquate. Ainsi, si l'on recherche un objet semblable dans les deux images, de mêmes taille et aspect mais pas à la même place ou n'ayant pas la même orientation, nous définirons une **transformation *rigide***, qui est le type de transformation le plus simple. Elle n'agit pas sur la forme des objets rencontrés mais se contente d'effectuer des opérations matricielles de base comme la rotation et la translation. Si les objets ont la même forme mais pas la même taille, une **transformation *affine*** sera nécessaire. Dans ce type de transformation, la notion de mise à l'échelle est ajoutée. Une droite reste une droite mais elle peut être agrandie ou rétrécie suivant les cas. Si les objets que l'on cherche à mettre en correspondance sont présentés sous un angle différent dans les deux images, nous utiliserons une **transformation *projective***. Enfin, la transformation nécessaire peut être beaucoup plus complexe. Il se peut que l'on observe une forme sous l'action de forces intérieures ou extérieures, subissant des

déformations calculables d'une image à l'autre. Nous aurons alors recours aux **transformations *élastiques*** pour le recalage de telles images. Ce sont de loin les plus compliquées à implémenter.

Dans la plupart des cas, ces transformations peuvent être décrites sous une forme paramétrique. Ainsi, une rotation/translation dans le plan est régie par trois paramètres[1] , une transformation *affine*, dans le plan, par six paramètres[2] et les autres types de transformations par un nombre de paramètres beaucoup plus élevé. [T-Thi97].

Voici, ci-dessous, une exemple de transformations rigides: l'image reste inchangée:

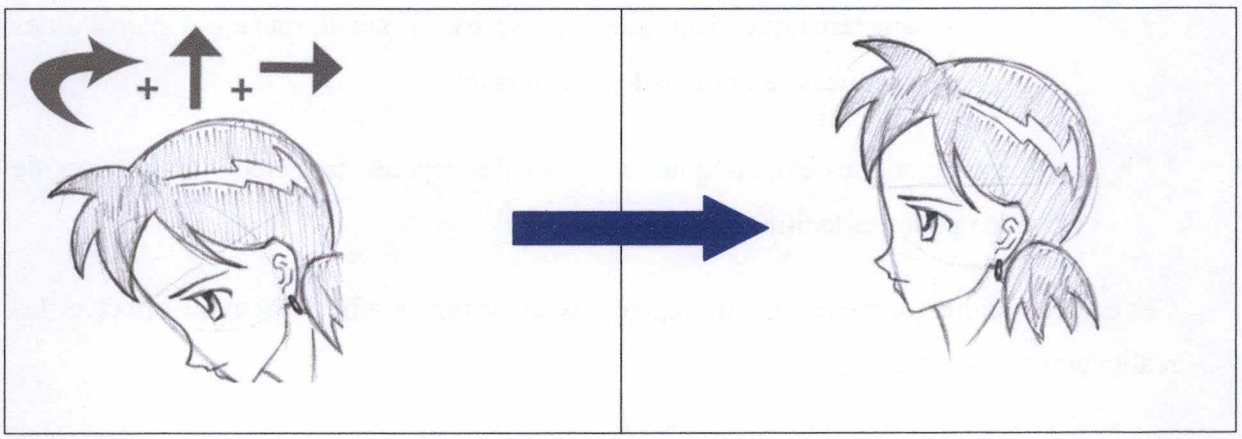


Figure 20 : Exemple de transformations rigides

Qui nous donne une matrice de ce genre:

$$\begin{pmatrix} \text{Cos } 30° + 20 & \text{- Sin } 30° \\ \text{Sin } 30° & \text{Cos } 30° \text{ - } 35 \end{pmatrix}$$

[1] Le déplacement en X et en Y ainsi que l'angle de rotation

[2] le déplacement en X et en Y, l'angle de rotation, les paramètres de la matrice de cisaillement (X'=X+BY ; Y'=AX+Y) et un facteur d'échelle.

Un exemple de transformation semi-rigide. L'image est déformée mais, dans ce modèle, une droite reste une droite. Elle sera juste redressée afin de rendre sa forme normale à l'image.

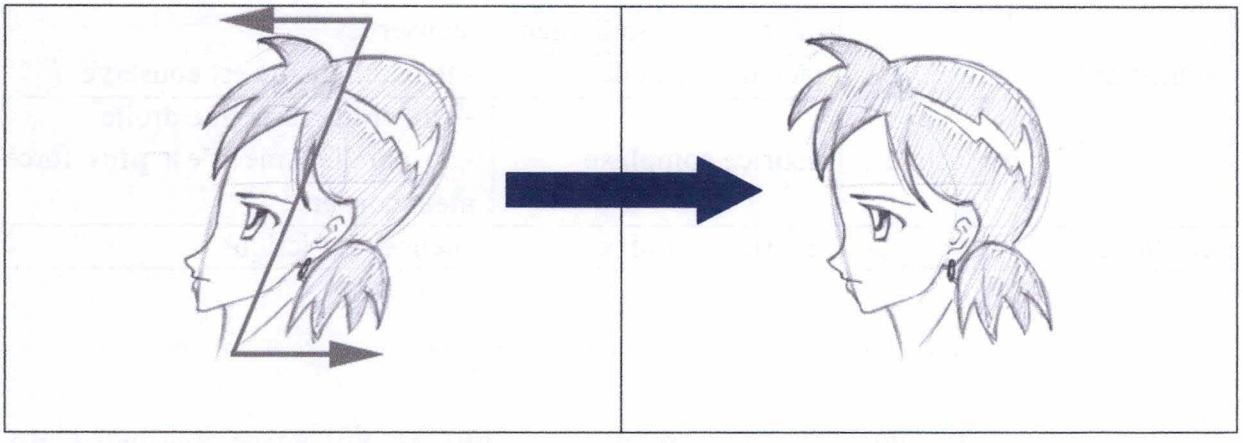


Figure 21 : Exemple de transformation semi-rigide dite "projective"

Et enfin, un exemple de transformation élastique. Dans ce type de transformation, tout est permis. Ici, par exemple, l'image va subir une sorte de rotation interne, une torsion, afin de retrouver son aspect original. Cette technique est sans doute la plus difficile à mettre au point et les matrices générées peuvent être assez complexes.

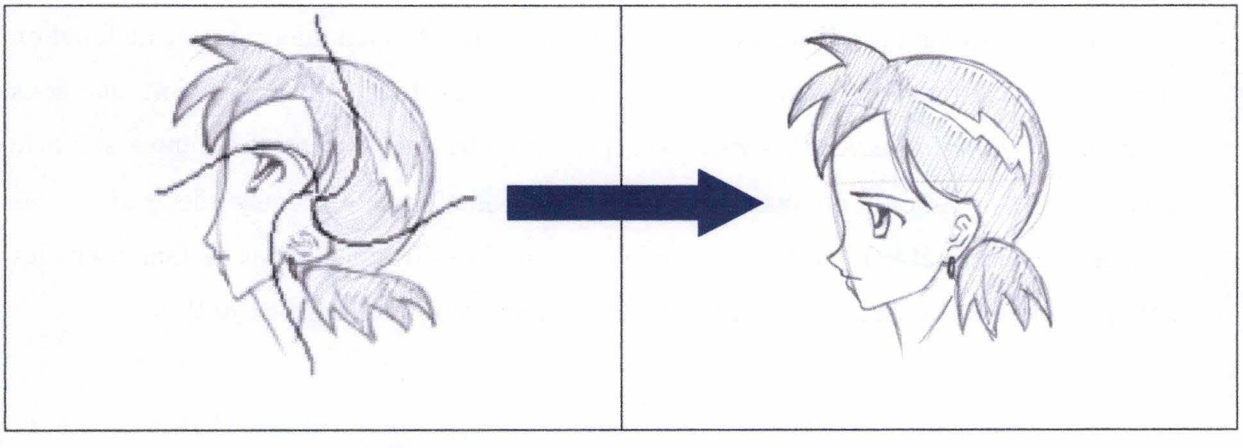


Figure 22 : Exemple de transformation élastique

Pour synthétiser, nous pouvons jeter un œil sur le tableau récapitulatif de la page suivante, montrant les différents types de transformations, classées selon leur degré de difficulté.

transformation		paramètres	propriétés après transformation
rigide		vecteur de translation	- la distance entre deux points est conservée
		angle de rotation	
semi-rigide	affine	vecteur de translation	- une droite reste une droite - les rapports de longueur sont conservés - le parallélisme est conservé
		angle de rotation	
		matrice de cisaillement	
		facteur d'échelle	
	projective	matrice complexe	- une droite reste une droite - le parallélisme n'est **plus** forcément conservé
élastique		matrice complexe	aucune restriction

Tableau 0: Classification des différents types de transformation. [M-Zuy03]

3. Nous venons de voir qu'il est possible de décrire les différentes transformations nécessaires au recalage sous une forme paramétrique. La prochaine étape qui en découle logiquement et de rechercher la valeur que prendront ces différents paramètres. Deux méthodes existent pour effectuer cette tâche. La première est d'exécuter un **algorithme déterministe** capable de déterminer les paramètres après analyse des images à corégistrer. Cette méthode est généralement peu employée dans les algorithmes de corégistration et nous n'irons donc pas plus loin dans sa présentation. La seconde méthode est d'employer un **algorithme itératif** qui déterminera les paramètres au fur et à mesure, en minimisant une fonction estimant les différences entre les deux images ou en maximisant une fonction mesurant le degré de similarité entre ces images. [W-CorImg]. Cette fonction, que nous nommerons ***fonctionnelle d'appariement*** peut prendre des formes différentes selon le type de transformation à effectuer. Le choix de la fonctionnelle adéquate déterminera les chances de succès et la rapidité de la corégistration. Les différents types de fonctionnelles sont présentés en figure 23. Pour plus d'informations, se référer à [M-Sey02].

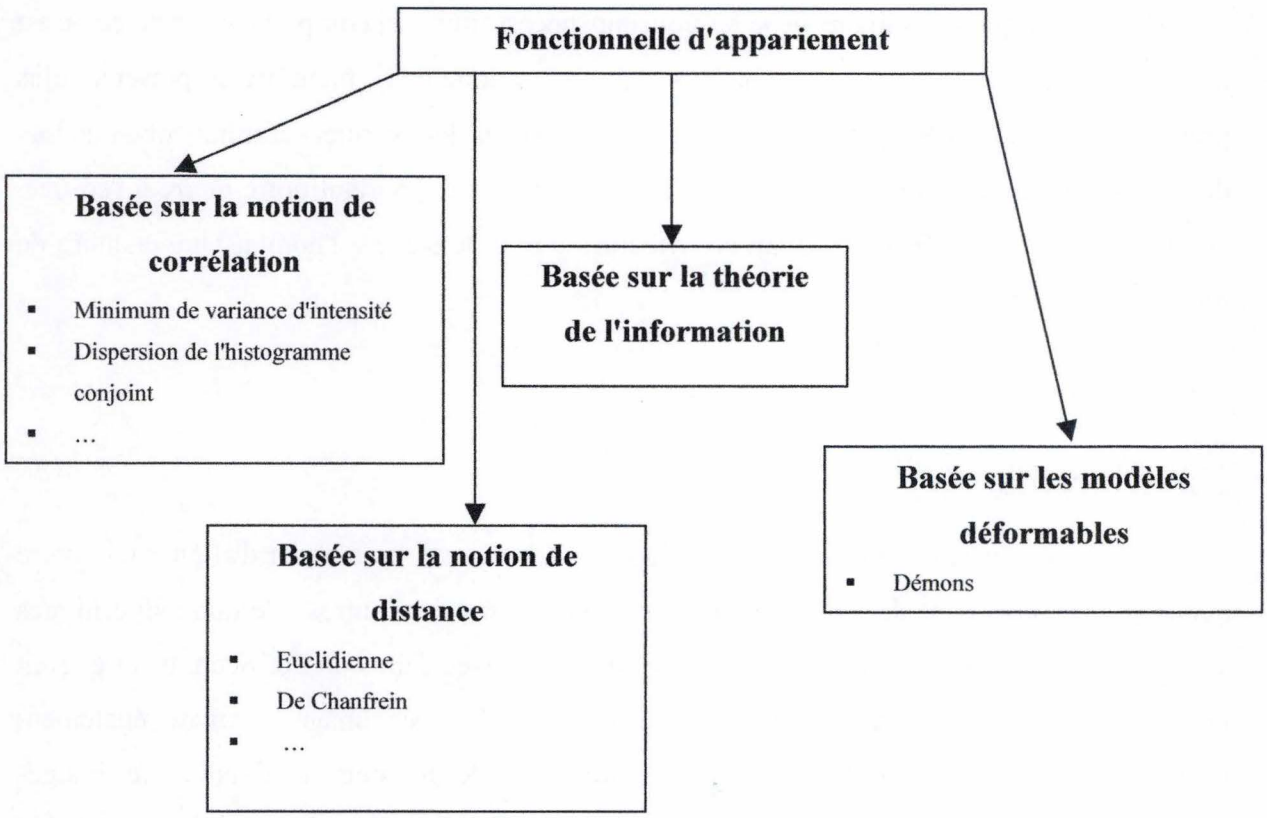


Figure 23 : Liste des différentes fonctionnelles d'appariement

4. Enfin, maintenant que nous savons quelles transformations effectuer et quelle est la fonctionnelle qui nous guidera le mieux dans la recherche des paramètres, il nous reste à trouver un extremum global de la fonctionnelle adoptée. Si la transformation ne dépend que d'un paramètre, nous pouvons envisager une recherche exhaustive en testant toutes les valeurs possibles[3]. Mais malheureusement, ceci est rarement le cas et il faut donc se munir d'algorithmes d'exploration des solutions sur l'espace de recherche. Ces algorithmes portent le nom de ***procédures d'optimisation*** et constituent l'ultime étape de corégistration. Pour exemple, nous pouvons citer les algorithmes génétiques et la méthode de descente du gradient. Pour plus d'informations sur ces techniques, le lecteur est invité à consulter [M-Sey02].

[3] Dans le cas d'une simple rotation par exemple.

Maintenant que nous avons posé les concepts nécessaires à la compréhension de ce qu'est la corégistration, nous pouvons entrer dans la partie pratique de ce mémoire en présentant les problématiques soulevées par cette technique, en décrivant les premiers résultats obtenus lors des travaux précédents [M-Sey02] et [M-Vae03] et enfin en apportant notre pierre à l'édifice, en présentant les modifications et ajouts effectués durant le stage à l'hôpital Universitaire de Mont-Godinne.

2.2 Objectifs

Il n'existe pas, en corégistration, d'algorithme universel permettant d'aligner n'importe quelle image provenant de n'importe quelle modalité avec une autre. Certains algorithmes sont orientés vers la corégistration des images de l'IRM avec celles du Pet-Scan, d'autres sont plus adaptés à la corégistration entre CT-Scan et IRM. Des images peuvent également provenir d'un même patient, d'une même modalité mais à des moments différents. Ces images doivent aussi être corégistrées car il peut y avoir des différences de placement du patient d'un examen à l'autre. Toutes ces raisons font qu'il existe de meilleurs packages de corégistration que d'autres, suivant les utilisations que l'on veut en faire. Le médecin n'ayant pas les connaissances informatiques nécessaires à la manipulation de tous ces programmes, la plate-forme de corégistration est une réalisation d'une grande importance. Elle doit permettre de choisir automatiquement le package le mieux adapté et les paramètres optimaux, en fonction des caractéristiques extraites des images, sans que le médecin n'ait à intervenir ou, le cas échéant, en guidant du mieux possible ce dernier. Nous voulons donc passer d'une manière procédurale de résoudre le problème de la corégistration à une manière plus déclarative, en essayant de le modéliser, de sorte qu'ils puisse être résolu automatiquement par la plate-forme, en proposant, exécutant et enchaînant les traitements adéquats.

Un autre objectif de la plate-forme est de pouvoir présenter les images rendues par les programmes de corégistration. Ceux-ci ne sont pas dotés de mécanismes d'affichage des résultats. Ils se posent généralement comme de simples programmes en ligne de commande auxquels on fourni un fichier contenant une image médicale en entrée et qui nous redonnent, en sortie, un autre fichier contenant l'image alignée. La visualisation des images coregistrées est d'une grande importance et englobe plusieurs techniques variées, allant du parallel display, pour la plus simple, a la superposition par bandes de couleurs ou par transparence.

Certaines techniques seront présentées au point 4.3.4 dans ce mémoire. Pour plus d'informations, le lecteur peut consulter [M-Van98].

Nous allons maintenant présenter l'existant de la plate-forme. Ce qui est développé dans [M-Sey02] et [M-Vae03] et qui nous a été laissé.

2.3 Description de l'existant

Le travail réalisé au cours de ces quatre mois de stage n'est pas le seul à avoir été effectué sur le sujet. Il s'inscrit dans une continuité de recherches menées par des étudiants d'années précédentes et marque la première réalisation **concrète et utilisable** de ce projet. Les principes de la corégistration ayant été exposés, nous allons parler de l'existant, en décrivant les travaux réalisés dans [M-Sey02] et [M-Vae03], avant de présenter notre propre réalisation, en mettant en évidence les différences et les ajouts réalisés par rapport à l'existant.

Première ébauche de la plate-forme:

Le première étude est celle retrouvée dans [M-Sey02]. Elle apporte une première ébauche de ce que doit être la plate-forme de corégistration multimodalités dans sa description orientée objet. Ce modèle se décompose en quatre couches, représentant les différents niveaux d'abstraction. Ces niveaux vont permettre de résoudre les problèmes relatifs à la découpe de la corégistration discutés en 2.1.2.

Les bases de ce modèle sont les suivantes:

- Dans la première couche, on retrouve l'image médicale et les différents objets qui la composent, notamment les "*Raw Data*"(ou données représentant les pixels) et ses données contextuelles. Comme nous l'avons précisé en 1.1.2, une image médicale est composée d'une, voire de plusieurs images numériques et d'informations textuelles sur le patient, l'examen réalisé, la modalité utilisée,... Un ensemble de classes est donc prévu afin de mémoriser ces données. En page suivante, vous pourrez découvrir la structure proposée dans [M-Sey02] et reprise dans [M-Vae03] pour une première implémentation. Nous verrons par la suite que cette couche a été appréhendée d'une manière autre dans ce travail que ce qui est proposé dans [M-Sey02] pour des raisons de facilité avec les algorithmes de corégistration.

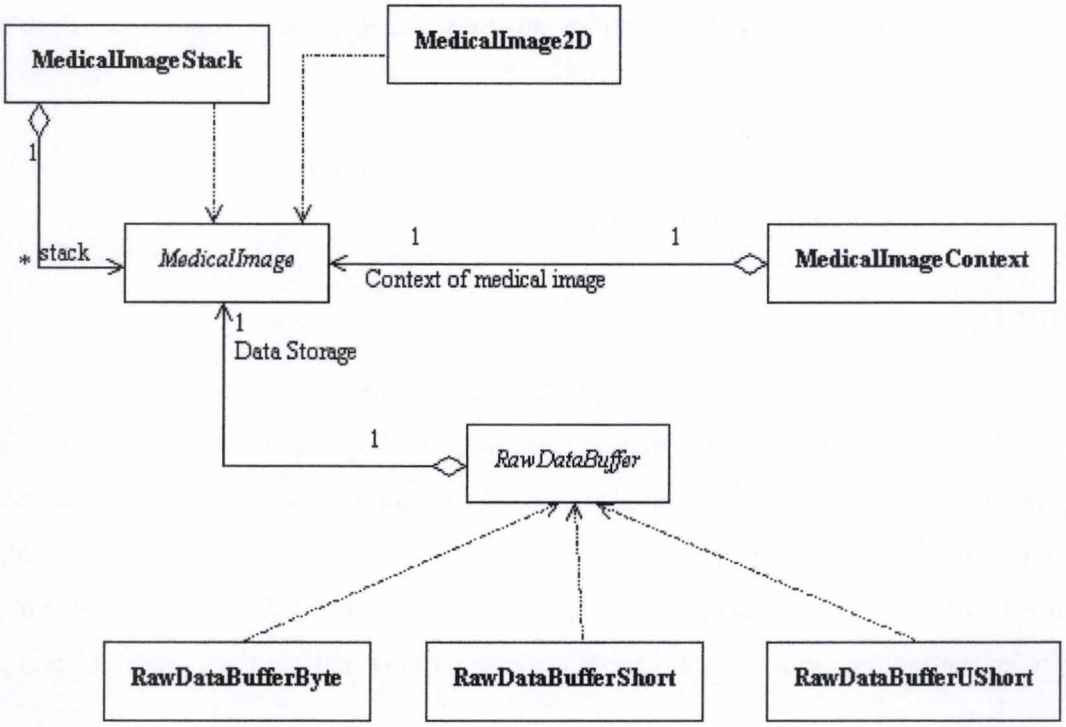


Figure 24 : Modélisation la couche 1 réalisée par Stéphane Seynave

La première partie de cette couche se compose des classes nécessaires à la mémorisation des données textuelles de l'image. Nous la nommerons "contexte médical de l'image". Nous ne nous y intéresserons que très peu dans ce travail car l'attention n'a pas été portée sur la gestion des données patient, dans un premier temps, mais bien à la mise en place d'une plate-forme permettant de corégistrer et de présenter les images. La seconde partie concerne la mémorisation des données brutes composant l'image. Ces données pouvant être de plusieurs types (byte, short, unsigned short), plusieurs classes sont modélisées afin de contenir un type de données particulier[4].

- Dans la deuxième couche, on découvre quatre subdivisions définissant les quatre grandes dimensions du recalage présentées en 2.1.2. Pour rappel, ces dimensions sont: les caractéristiques des images, la transformation, la fonctionnelle d'appariement utilisée et les procédures d'optimisation utilisées.

[4] Les données brutes des images médicales peuvent être de plusieurs types différents (byte, short, unsigned short,...)

La modélisation de chacune de ces classes est la suivante:

- Pour l'espace des caractéristiques, seules les caractéristiques intrinsèques ont été modélisées. Ce sont donc les caractéristiques disponibles directement dans les images. La figure suivante illustre la représentation adoptée:

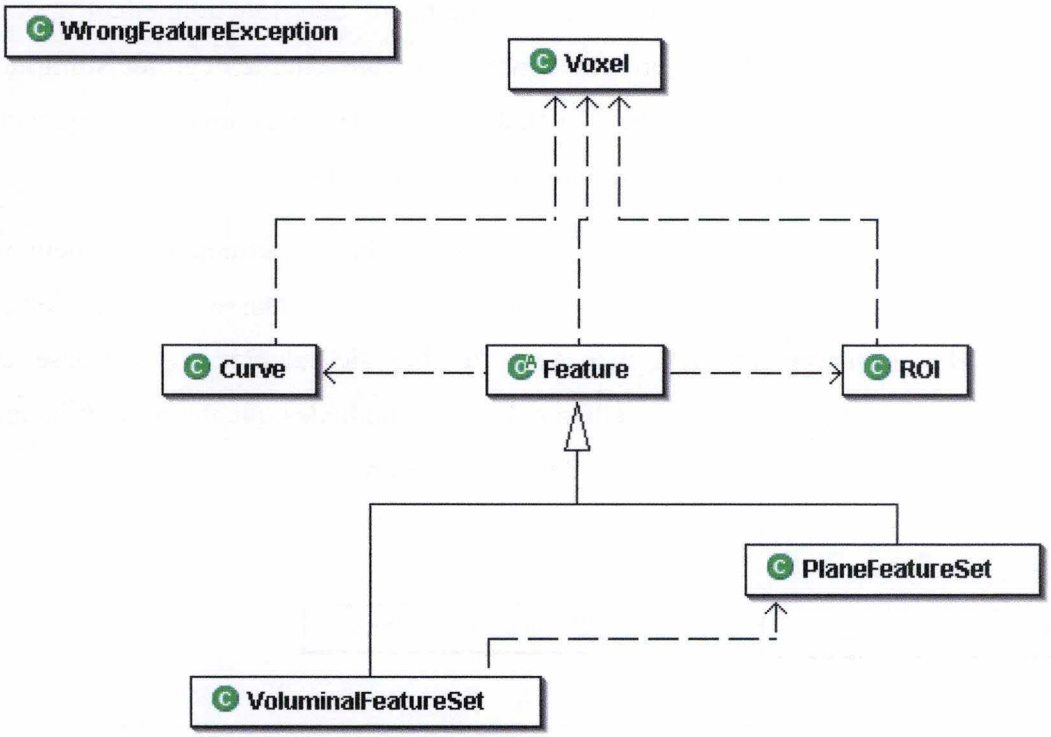
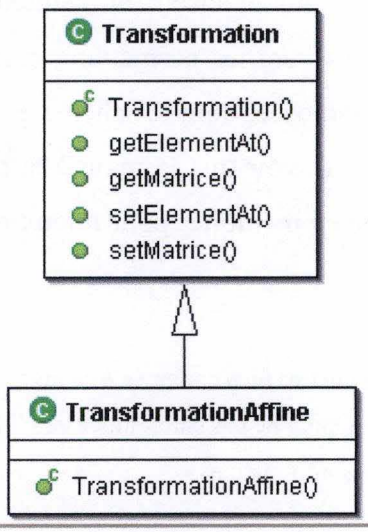


Figure 25 : Classes de gestion des caractéristiques de l'image

- Les transformations, présentées au point 2 du 2.1.2 sont modélisées par une classe "Transformation" disposant d'une matrice et des méthodes d'accès à ses éléments.



La classe "AffineTransformation" représente une transformation affine. Elle hérite de "Transformation".

Figure 26 : Modélisation des transformations

- La fonctionnelle d'appariement, qui est donc la fonction qui permet de minimiser les différences entre deux images ou de maximiser la ressemblance[5].
- La méthode d'optimisation est relativement peu développée. La classe abstraite "OptimisationMethod" se charge de la représenter. Elle définit la méthode "findTransformation" qui prend en argument une fonctionnelle d'appariement, permettant ainsi de mesurer, au cours de la recherche, le degré de similitude entre les deux images et les caractéristiques issues des deux images à corégistrer. Le résultat sera la transformation optimale. [M-Vae03].

- Au troisième niveau dit de *Stratégie*, plusieurs es classes permettent de mémoriser les différentes stratégies de corégistration disponibles. La classe "RegistrationContext" permet de conserver une référence vers la stratégie adoptée. La classe abstraite RegistrationStrategy sert, quand à elle à définir les méthodes que devront obligatoirement implémenter l'ensemble des stratégies de corégistration.

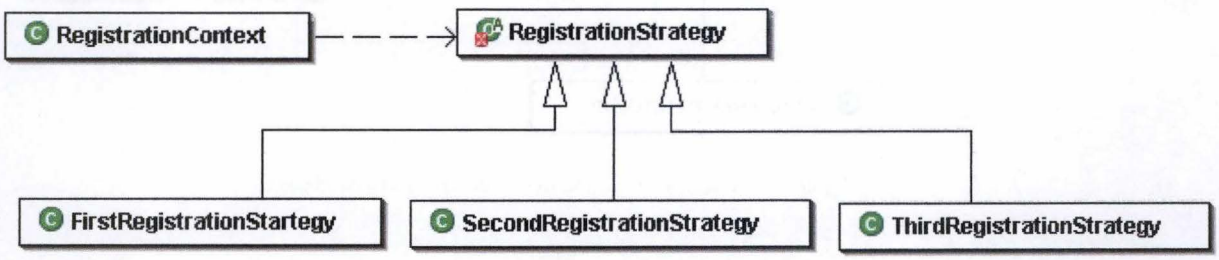


Figure 27 : Couche 3: Stratégies de corégistration

- Enfin, au quatrième et dernier niveau, on retrouve une classe de haut niveau permettant de gérer l'ensemble de la problématique : le client ou, plus précisément, l'éditeur spécialisé, qui n'est en fait qu'une interface graphique d'entrées / sorties permettant, entre autres, de lancer la tâche de corégistration... L'architecture proposée, dont le schéma est présenté en page suivante, est difficile à imaginer sans vraiment avoir mis la main à la pâte et elle a été remodélisée dans son entièreté, dans la suite de ce mémoire, pour mieux répondre aux besoins des utilisateurs.

[5] La représentation de cette couche faite dans [M-Vae03] n'a pas été très bien comprise à ce niveau et ne semble pas adéquate... nous ne la décrirons donc pas dans cette section, pour ne pas embrouiller les choses, et invitons le lecteur à consulter [M-Vae03] pour de plus amples informations.

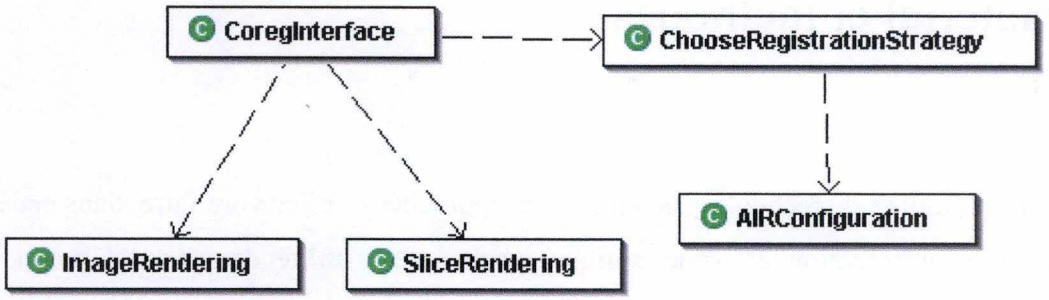


Figure 28 : Modélisation de la couche 4 réalisée dans [M-Vae03]

La classe CoregInterface est le point d'entrée du programme. Elle représente l'interface principale. De cette interface, il est possible d'ouvrir, fermer et afficher des images médicales. La classe ChooseRegistrationStrategy permet de choisir la stratégie de corégistration utilisée, autrement dit, les algorithmes qui vont être utilisés (dans le cas présent, rappelons que seul AIR a été implémenté). Enfin, les classes ImageRendering et SliceRendering permettent de gérer l'affichage des images médicales pour CoregInterface.

Cette ébauche d'architecture constitue une base solide pour le projet à réaliser et est implémentée dans sa première version dans [M-Vae03]. Nous verrons cependant, dans la suite de ce mémoire, que quelques aménagements devront être concédés à l'architecture de base afin de pouvoir rendre le programme utilisable et plus souple par la suite. La couche 4, telle qu'elle a été modélisée dans [M-Vae03], s'est révélée inadéquate, et est entièrement refondue dans le chapitre 4.3. La couche 1 est partiellement revue dans le chapitre 4.1.1, pour des raisons de facilité d'interfaçage des algorithmes de corégistration. Enfin, la couche 2 se verra légèrement complétée pour l'intégration du premier algorithme de corégistration. La couche 3 ne sera pas du tout abordée dans le reste de ce mémoire car le but est, dans un premier temps, de pouvoir intégrer une seule stratégie de corégistration. Il ne faudra donc pas effectuer de choix.

Notons enfin que nous pourrons également nous référer à [M-Van98], qui nous a apporté quelques bonnes idées à appliquer pour la visualisation des images corégistrées.

L'existant ayant été présenté, il nous reste maintenant à décrire les technologies utilisées avant de pouvoir présenter notre propre réalisation.

3 Matériel et méthodes

Bon nombre de technologies ont pu être approchées. Elles vont faire, dans cette partie, l'objet d'une présentation et d'une critique quant à leur utilité dans un tel projet. Nous séparerons les technologies selon qu'elles sont utilisées pour la partie corégistration ou visualisation de la plate-forme.

La première chose à faire est de présenter le langage de programmation utilisé et de montrer les avantages qu'il apporte dans un travail comme celui-ci.

3.1 JAVA

L'utilisation de java dans ce projet n'est pas un choix personnel et il n'est donc pas possible de pouvoir le justifier sous toutes ses coutures. Néanmoins, en vu de l'expérience acquise au cours de ce stage, Il est aisé de se rendre compte des nombreux avantages offerts par cette excellente plate-forme de développement. Une présentation de ce qu'est JAVA est considéré comme inutile dans un travail comme celui-ci. C'est pourquoi nous allons directement passer à l'énumération des avantages offerts et à la pertinence de l'utilisation de ce dernier dans un travail comme celui-ci.

Les raisons les plus probantes pour lesquelles JAVA a sans doute été choisi sont:

- Sa modularité: Java est **le** langage orienté objet par excellence (même si récemment, de nouvelles technologies telles que C#[6] font leur apparition).
- Sa popularité: il y aurait environ trois millions de développeurs qui utiliseraient java. L'avantage que ceci apporte est qu'il y a certainement quelqu'un, quelque part, qui sera apte à répondre à vos questions.

[6] Le langage C# créé par Microsoft dans le but de concurrencer Java, est devenu un standard approuvé par la spécification ECMA-334 de l'association européenne ECMA (European Computer Manufacturer's Association) en décembre 2001. Il permet de développer des applications Windows et Web en se basant sur l'architecture .NET. Ce langage a été conçu dans le but d'être indépendant de la plate-forme et de l'environnement d'exécution. [http://www.laltruiste.com/document.php?compteur=4&rep=12&evolution=2]

- Sa portabilité: aucun changement du code n'est nécessaire à l'utilisation des programmes rédigés en java sur une plate-forme ou sur une autre.

- Sa puissance: certains pourraient être surpris de retrouver une telle qualité dans un langage réputé lent. Tout dépend aussi de ce que nous entendons par *puissance.* Si nous parlons de puissance de modélisation et de conception, JAVA nous offre des mécanismes très précieux, nécessitant nombre de lignes de code dans un langage comme le C++ par exemple. Prenons la réflexivité par exemple, qui fera d'ailleurs l'objet d'une présentation au point 3.4.1. Ce mécanisme peut se révéler d'une grande utilité pour certaines tâches. Si, par contre, nous parlons de la puissance de calcul, contrairement à ce que beaucoup pensent, JAVA est un langage qui peut se montrer extrêmement rapide et même, dans certains cas, dépasser le leader : le C++, comme le montre la figure 29. Ceci, grâce aux optimisations qu'il est possible de réaliser avec la machine virtuelle.

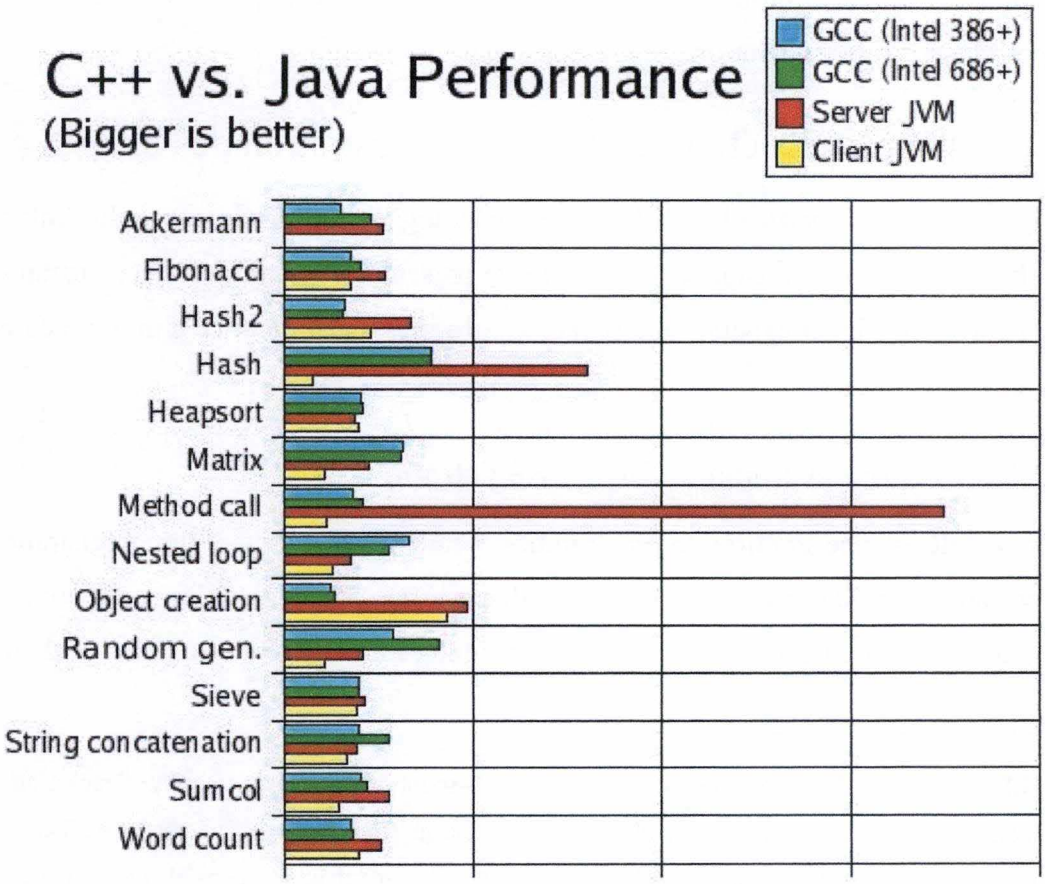


Figure 29 : Performances de java par rapport au C++ (http://www.kano.net/javabench/data)

Sa lenteur provient entre autres de la gestion des composants graphiques SWING et AWT. Mais, là encore, avec un peu d'astuce, il est possible d'optimiser tout cela.

- JAVA est entouré d'un bon nombre d'outils d'aide au développement, gratuits et d'une grande utilité. Pendant le stage, par exemple, les outils NetBeans® et Eclipse® ont été largement utilisés et, même s'ils nécessitent un certain délai d'adaptation, ils ont grandement accéléré le développement de l'application.

- Enfin, un des objectifs du projet est d'intégrer la plate-forme à un environnement existant nommé *TELEMIS®* [7]. Cet environnement étant lui-même développé en JAVA, l'intégration n'en sera que facilitée.

Il serait facile de continuer à faire l'éloge et à énumérer les avantages de JAVA mais ceci n'est pas le but. Nous allons maintenant passer à la description des technologies (gratuites elles aussi) utilisées lors du développement, pour la partie visualisation des images et pour la partie corégistration.

3.2 Technologies étudiées pour la corégistration des images

Dans cette partie, les différentes technologies approchées pour la réalisation de la partie corégistration de l'application vont être présentées. Certaines de ces technologies sont les algorithmes de corégistration eux-mêmes, d'autres constituent ce dont on a eu besoin pour les faire fonctionner.

3.2.1 AIR pour Automated Image Registration

AIR est une librairie de programmes autonomes permettant la corégistration d'images provenant de modalités différentes. On dispose des sources et l'on peut compiler chaque programme séparément afin de les lancer en ligne de commande. Ce package de programmes

[7] TELEMIS® est un des leaders dans le domaine de la distribution d'images médicales. Grâce à ses compétences de pointe en compression d'images et en sécurité, l'imagerie peut être partagée efficacement à l'intérieur et à l'extérieur de l'hôpital. Telemis est une solution modulaire et extensible qui s'intègre dans un Système d'Information Hospitalier.
[http://www.sectors.walloniaexport.be/nitc/SOCIETES/FICHEINFEN69F481.HTM]
Pour plus d'informations, leur site web est disponible à cette adresse: http://www.telemis.com

a été validé pour la corégistration d'images provenant de la résonance magnétique et du pet-scan mais peut être utilisé avec d'autres modalités.

Il existe plusieurs programmes livrés avec AIR. Outre le programme de corégistration principal, on peut citer *Resize* qui permet de redimensionner une pile d'images dans toutes les directions de l'espace (X,Y et Z), *Zoomer*, qui permet d'obtenir des voxels cubiques et beaucoup d'autres encore, qui permettent généralement d'effectuer des optimisations sur les images à corégistrer (filtres gaussiens permettant de nettoyer une image du "bruit",...) .

Le format d'images reconnu par AIR est le format ANALYSE, développé à la clinique Mayo[8].

Le fonctionnement standard d'un programme AIR est le suivant :
On lance le programme en ligne de commande, en lui donnant les paramètres souhaités.
Par exemple, "resize params input ouput" lancera le programme resize avec les paramètres donnés, soit l'image « input » en entrée et l'image "output" en sortie.

La corégistration automatique se fait en deux étapes, dans AIR.

1. Dans la première étape, que l'on nomme l'alignement, l'image à corégistrer est comparée à l'image maître. L'image à corégistrer va faire l'objet de transformations successives, de manière à minimiser les différences avec l'image maître. Une fois cela réalisé, une matrice contenant les transformations nécessaires à la corégistration sera sauvegardée dans un fichier de format propre à AIR. Cette étape correspond donc à l'étape 4 de la corégistration d'images. L'étape 3 est fixée dans l'algorithme et les deux premières étapes doivent être spécifiées par l'utilisateur à l'aide de paramètres. L'utilisateur doit donc pouvoir se rendre compte du type de transformation à effectuer (rigide, affine, projective,...) en fonction des images qu'il veut corégistrer. Ce qui montre déjà un des problèmes qui devra être résolu dans la plate-forme.

[8] Ceci constitue déjà une premier défi pour notre future application car celle-ci tournera avec des fichiers au format DICOM.

2. La seconde étape ne consiste qu'en l'application de la matrice sauvegardée à l'image à corégistrer. Une nouvelle image sera donc créée à partir de ces transformations.

Un des buts de ce mémoire, est de trouver un moyen d'intégrer, d'une manière complètement transparente pour l'utilisateur, AIR, dans un environnement java. Pour ce faire, il serait judicieux de

- Ne plus passer en ligne de commande, mais de disposer d'une interface propre et soignée faisant appel aux différents programmes.

- Permettre d'utiliser d'autres images que les fichiers ANALYSE et, dans un premier temps, les fichiers DICOM.

3.2.2 JNI pour Java Native Interface

JNI est l'abréviation de Java Native Interface. Cette technologie permet d'utiliser du code natif (écrit en C ou en C++) dans un programme JAVA. [W-JNI]. Elle peut s'avérer utile à plus d'un titre ; certains algorithmes sensibles peuvent être écrits en C[9] ou certains composants
éprouvés peuvent être récupérés et insérés dans un projet JAVA. Dans notre cas, nous sommes friands de ces deux avantages car les algorithmes utilisés lors de la corégistration tridimensionnelle sont souvent complexes et fort demandeurs de ressources et, plus particulièrement, celui qui nous intéresse ici (AIR), est écrit en C. Néanmoins, l'utilisation de JNI n'apporte pas que des avantages et un de ses inconvénients majeurs est qu'elle annihile la portabilité du programme. Il faut recompiler la partie native pour chaque plate-forme utilisée.

La mise en œuvre d'une application faisant intervenir JNI peut se décomposer en plusieurs étapes distinctes:

1. Premièrement, il faut déclarer l'utilisation des méthodes natives dans le programme JAVA. Ceci se fait très simplement, en rajoutant l'acronyme "native" devant la déclaration d'une méthode java.

   Ex: public native void nativeProc();

2. Ensuite, il faut compiler cette classe comme à l'habitude.

3. Troisièmement, un utilitaire livré en standard avec JAVA, du nom de javah nous sert à générer les fichiers d'en-têtes (fichiers .h en C ou .hpp en C++) nécessaires à l'écriture du code natif.

4. L'écriture des méthode natives se fait en respectant strictement la signature de méthodes générées par javah.

[9] Nous avons vu en 3.1 que JAVA pouvait se révéler assez rapide, mais le graphique nous montre bien qu'il existe encore un net avantage à utiliser du code natif pour la manipulation de matrices.

5. Ce code natif est compilé sous la forme d'une librairie dynamique (dll pour Windows ou .so pour Linux) qui peut alors être chargé dans le programme JAVA lors de l'exécution de celui-ci.

JNI, outre la possibilité d'appeler du code natif en JAVA, permet de partager certaines zones mémoire, rendant accessible au C des tableaux créés en JAVA et vice versa. Il faut cependant faire preuve d'une grande prudence dans la manipulation de telles données car, le C étant beaucoup plus laxiste que JAVA, il est tout à fait possible de boycotter des espaces mémoire utilisés par JAVA.

L'utilisation de code natif en JAVA est relativement aisée **si et seulement si** l'on respecte scrupuleusement ces indications! La moindre fausse manœuvre viendra fausser l'ensemble de la procédure[10].

[10] Un exemple de code natif Intégré à un programme JAVA peut être trouvé en annexe II.

3.3 Technologies étudiées pour la visualisation des images

Les technologies présentées dans ce chapitre ont été approchées pendant la réalisation de l'application. Certaines sont employées dans la version finale, d'autres sont mises momentanément de côté car elles ne correspondent pas à l'utilisation que l'on veut en faire. Dans cette partie visualisation, les technologies utilisées font toutes partie des nombreux ajouts disponibles pour JAVA, dont on a parlé en 3.1.

3.3.1 JAI pour Java Advanced Imaging

JAI est une extension de la plate-forme JAVA conçue pour des applications de pointe en traitement d'images. [W-JAI]. En plus des traitements de base comme la lecture ou l'écriture de certains formats de fichiers, JAI permet d'effectuer nombre de traitements utiles sur les images qu'il manipule. JAI dispose également d'un système d'affichage des images qui lui est propre.

Cette description met évidemment JAI en avant plan pour notre application. Nous pourrions tirer avantage des outils que propose, en standard, cet API puissant. Ces outils vont du réglage automatique de la luminosité et du contraste, à l'application de filtres de traitement d'images, de génération automatique d'histogrammes et encore d'autres choses bien utiles.

Malgré tous les avantages qu'apporte l'API, il s'avère que JAI ne peut convenir pour un tel projet, du moins pour l'affichage des images avec une rapidité décente. Si JAI se révèle être d'une puissance extrême dans le traitement des images, la partie "affichage" n'est malheureusement pas à la hauteur, ou, du moins adaptée à notre projet. Il existe en effet très peu de moyens d'afficher une image au format JAI, il faut utiliser les outils fournis dans l'API, et ces outils sont très restrictifs et extrêmement peu paramétrables.

3.3.2 Java 3D

JAI ne correspondant pas, il faut se tourner vers une autre technologie pour afficher nos images. Le choix suivant est Java3D. Cela peut paraître bizarre mais plusieurs raisons font de Java3D un outil prometteur.

Java3D est une API JAVA dédiée à la manipulation d'images tridimensionnelles. La force de Java3D est qu'il s'appuie sur des technologies éprouvées telles que DirectX® et OpenGL®. Ces technologies, permettent d'utiliser la puissance des cartes graphiques pour

l'affichage des objets 3D et 2D, laissant le processeur disponible pour d'autres tâches que l'affichage.

Bien que fort alléchante encore, cette technique s'avère apporter plus d'ennuis que d'avantages. Il faut en effet plaquer l'image sur une face, comme une texture, et recalculer la position de cette texture à chaque manipulation de l'image[11]... pas très élégant! Java3D est donc, lui aussi, abandonné. Il ne faut toutefois pas négliger les possibilités de ce dernier. Il peut, lui aussi, être utile sur bien des points. Par exemple, pour la programmation d'un algorithme de reconstruction des organes en 3D.

3.3.3 Java2D

Pour finir, JAI et Java3D n'étant pas convaincants, il ne reste plus que Java2D. Au départ délaissé car imaginé trop simpliste mais, bien utilisé, se montrant d'une efficacité redoutable. Ce dernier se voit crédité d'une foule d'outils potentiellement intéressants et faciles à utiliser (antialiasing, interpolation, radial blur,...). C'est donc sur lui que le choix s'est porté et dont la présentation va pouvoir être un peu plus développée dans la suite de ce chapitre.

Pour présenter cette API, nous allons tout d'abord expliquer le principe d'affichage d'une image en java.

Le moyen le plus simple, pour afficher une image en JAVA, est de surcharger la méthode paint(). Cette méthode fait partie de tout composant graphique de base présent dans AWT ou SWING. Elle est appelée chaque fois que le composant doit être redessiné (par exemple lors d'un redimensionnement de l'application ou d'un appel implicite à repaint(). La signature de cette méthode est la suivante:

```
void paint(Graphics g);
```

Comme il est précisé ci-dessus, c'est le système qui appelle la méthode, jamais le programmeur (ou presque jamais, dira-t-on)! S'il veut faire un appel explicite à celle-ci, il se

[11] Java3D est fait pour afficher des scènes en 3D, pas pour afficher des images 2D

servira de la méthode **void repaint();**. On peut observer que paint prend un argument en paramètre. Cet argument est du type **Graphics**. L'objet de type **Graphics** passé en paramètre servira à dessiner sur le composant concerné. Illustrons ceci par un exemple pratique:

Imaginons que nous voulions dessiner un carré sur une JPanel (composant graphique de java). Nous étendons ce composant afin de pouvoir overrider la méthode paint :

```
public class MyDrawingPanel extends JPanel
{
        ...
        public void paint(Graphics g)
        {
                super.paint(g);
                g.drawRect(10,10,50,50);
        }
        ...
}
```

La méthode "paint" a donc été overridée. Il avait été dit qu'on ne ferait jamais appel à paint directement, pourtant, déjà ici, on fait un appel explicite… mais cet appel se fait sur le paint de la classe mère. Il va simplement servir à redessiner le composant comme il doit se faire. Si l'on ne fait pas cet appel, la fenêtre n'est jamais redessinée et à chaque déplacement/redimensionnement, elle se détériore un peu plus. La seconde méthode nous sert à dessiner un carré à la position (10,10) d'une longueur de coté de 50 pixels. Les objets de type **Graphics** nous permettent donc de dessiner plusieurs types de primitives (cercle, carré,…). Ceci ne sera pas intéressant pour ce travail (dans un premier temps). Seule la méthode permettant de dessiner une image nous sera utile. Cette méthode se nomme **drawImage** et a le prototype suivant :

Public void drawImage(java.awt.image.Image,int,int);

Le principe utilisé, pour afficher nos images, est de construire ces dernières à partir des données brutes lues dans les fichiers DICOM, dans un format que drawImage reconnaît afin que ce dernier puisse les afficher.

Jusqu'ici, nous avons vu comment afficher une image à l'aide du mécanisme standard de JAVA, ceci fonctionne très bien mais l'on peut faire mieux avec un tout petit effort, en utilisant Java2D. Java2D se base sur le même principe et, pour l'utiliser, il nous suffit de réaliser un cast de l'objet Graphics vers un objet de type Graphics2D comme ceci:

```
public class MyDrawingPanel extends JPanel
{
        ...
        public void paint(Graphics g)
        {
                Graphics2D g2=(Graphics2D)g;
                super.paint(g2);
                g2.drawRect(10,10,50,50);
        }
        ...
```

Comme l'on peut s'en rendre compte, la surcharge de travail n'est pas énorme. Mais cette petite modification va permettre de se servir d'outils bien pratiques offerts par Java2D comme l'antialiasing, l'interpolation, pour un coût de programmation voisin de zéro!

3.4 Méthodes et techniques

La plate-forme de corégistration a été pensée dans le but d'être la plus évolutive et la plus modulaire possible. Il sera ainsi assez aisé de rajouter de nouveaux outils de visualisation ou de corégistration avec un minimum, voire sans effectuer la moindre modification du code original. Le grand principe utilisé pour parvenir à cette fin est la réflexivité. Certains problèmes ont également été rencontrés lors de l'implémentation de la plate-forme. Un des points sensibles fut la gestion des événements. Les techniques utilisées seront donc également présentées en

3.4.1 La réflexivité

La réflexivité est le principe par lequel il est possible de récupérer des informations sur les classes et sur les structures de données à partir de leur nom, pendant l'exécution du programme. C'est un mécanisme extrêmement puissant qui n'a pas d'équivalent dans les langages tels que le C ou le C++ et qui va être d'une grande utilité dans un projet comme celui-ci. Grâce à ce mécanisme, on pourra, par exemple, énumérer toutes les méthodes d'une classe et les utiliser par la suite. Et ceci, pendant l'exécution même du programme. Si l'on connaît le nom d'une classe particulière, on pourrait également l'instancier à partir de ce dernier. Ceci peut paraître assez flou d'un premier abord. Cependant, avec un petit exemple, tout devrait vite devenir très clair.

Imaginons un programme auquel il serait possible d'ajouter des mécanismes par un système de plugins. Chaque plugin peut être développé par n'importe qui et le programme original n'en a pas connaissance. La réflexivité pourrait être d'une grande utilité dans un pareil cas. On pourrait en effet tenir un fichier de configuration reprenant le nom des classes représentant les plugins à instancier par le programme. A son lancement, le programme pourrait parcourir la liste des noms contenus dans ces fichiers et instancier les classes en conséquence.

La figure 30 de la page suivante illustre ceci.

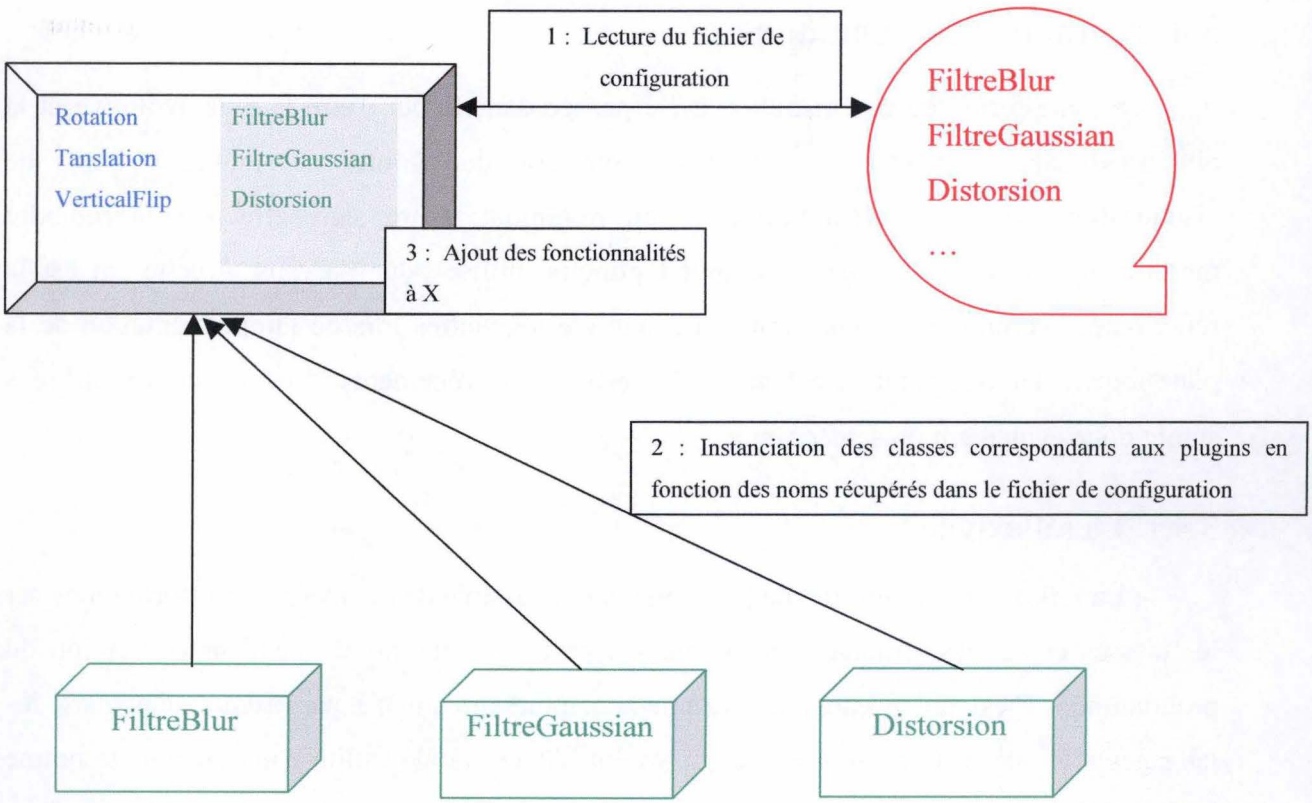


Figure 30 : Utilisation de la réflexivité pour créer un mécanisme de plugins

Dans cet exemple, le programme X lit le fichier de configuration. Il trouve alors les noms des plugins qu'il peut assimiler. Une fois le fichier lu, il trouve les fichiers contenant le code des plugins (en JAVA, le nom du fichier = le nom de la classe qu'il contient donc pas de problème) afin de pouvoir les instancier. Une fois les classes instanciées, les nouvelles fonctionnalités qu'elles contiennent sont ajoutées au programme.

Quels avantages y avait-il à tirer d'un tel outil dans notre plate-forme? Dès le départ, il a été précisé que cette plate-forme se voulait d'être évolutive au maximum. Donc, comme pour le programme X, nous allons lui permettre de s'étendre en évitant au programmeur une quelconque retouche dans le noyau de celle-ci. Il lui suffira de créer ses propres classes en implémentant une certaine interface, de modifier le fichier de configuration et sa contribution sera alors ajoutée automatiquement à la plate-forme.

3.4.2 La gestion des événements

La gestion des événements a été un point particulièrement sensible dans ce travail. Il fallait trouver un moyen élégant de transmettre l'information à travers la hiérarchie de composants. Les illustrations en sont nombreuses. Par exemple, dans le parrallel display, il faut répercuter l'événement du drag de la souris dans plusieurs panneaux d'affichage afin que les images défilent en même temps.

La première solution envisagée aurait pu être la plus simple et aussi la plus élégante. Dans AWT 1.0, la propagation des événements se fait assez facilement, en overridant une certaine méthode de façon à ce qu'elle renvoie la valeur false. Si tel est le cas, l'événement est propagé vers l'objet parent, celui d'où a été instancié l'objet actif. La signature de cette méthode est la suivante:

public boolean handleEvent(Event evt);

Elle est appelée chaque fois qu'un événement survient. Le programmeur doit alors analyser le type de l'événement (s'agit-il d'un bouton pressé ou d'un click de souris, de quel bouton s'agit-il?,...) pour savoir quoi faire. En fait, cela se passe de la même façon qu'en C ou C++ sous windows.

```
public boolean handleEvent(Event evt)
{
    switch (evt.id)
    {
            case Event.MOUSE_DRAG: do something...;break;
            case Event.MOUSE_UP: do something...;break;
            case Event.KEY_PRESS:
            case Event.KEY_ACTION:do something...;break;
    }
      return false;
}
```

Les événements AWT 1.0 sont traités par la méthode de transmission. Pour traiter les événements provenant d'un composant graphique, il faut donc créer une sous-classe de ce

dernier et overrider la méthode handleEvent(). Si, à la fin de cette méthode, on lui fait retourner la valeur "true", on dira que l'événement a été consumé et il ne sera donc plus transmis vers un autre composant. Par contre, si on lui fait retourner "false", l'événement sera propagé à travers toute la hiérarchie, jusqu'à ce qu'il soit consommé ou qu'il atteigne la racine.

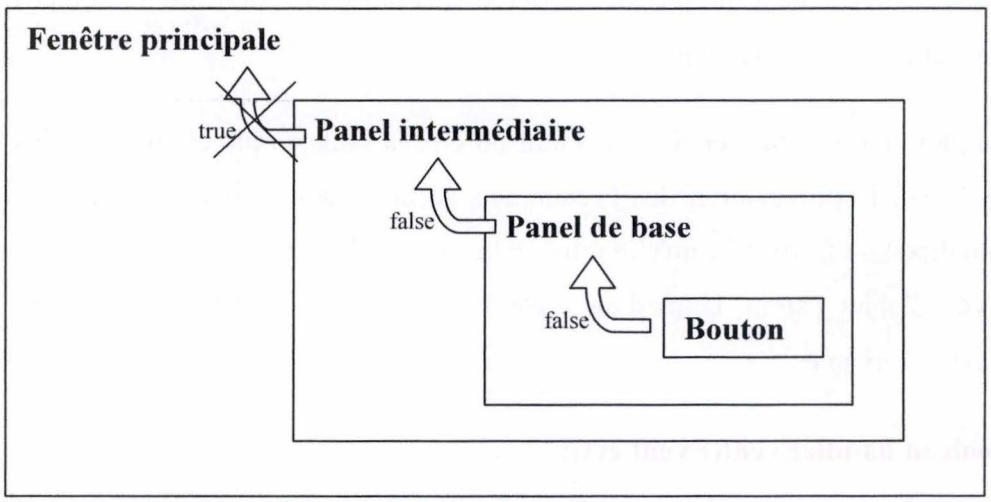


Figure 31 : Utilisation de la réflexivité pour créer un mécanisme de plugins

Dans l'exemple ci-dessus, on peut observer que l'événement est propagé du bouton jusqu'au panel intermédiaire. Il ne remontera pas jusqu'à la fenêtre principale.

Malheureusement pour nous, il y a un monde de différences entre la gestion des événements en AWT 1.0 et la gestion des événements dans les versions ultérieures et cette méthode, bien qu'elle soit toujours possible à mettre en œuvre, est dépassée et n'est plus utilisée.

Le modèle actuel de la gestion d'événements AWT se base sur le principe de délégation. D'après Sun®, les buts principaux qui ont fait naître ce nouveau modèle événementiel sont les suivants:

- Il est simple et facile à apprendre
- Il supporte une séparation claire entre les composants graphiques et l'application
- Il facilite la création de code, de gestion d'événements, robuste et moins exposé aux erreurs
- Il est assez flexible pour permettre la propagation des événements dans des modèles d'application variés.
- Il est compatible avec l'ancien modèle

Dans ce modèle, les événements sont maintenant organisés en une hiérarchie de classes événementielles. Il utilise des *sources d'événements* et des *écouteurs d'événements*.

Une source d'événements est un objet qui a la possibilité de déterminer quand un événement intéressant est survenu et d'en avertir les *écouteurs d'événements*.

Un écouteur d'événements est un objet qui implémente une *interface écouteur* spécifique. Un certain nombre d'interfaces écouteur sont définies où chaque interface déclare les méthodes appropriées pour une certaine classe d'événements. Il y a donc un certain nombre de classes implémentant des interfaces pour pouvoir traiter les événements. Nous avons, par exemple, une classe pour gérer les événements de la souris. Cette classe implémente l'interface correspondante et redéfinit les méthodes dont elle a besoin pour gérer ces événements.

Pour pouvoir être prévenu des événements qui surviennent, l'écouteur doit s'enregistrer auprès de l'émetteur pour le type d'événement qui l'intéresse. Une fois que l'écouteur est enregistré, la survenance d'un événement va automatiquement invoquer la méthode adéquate de l'objet écouteur.

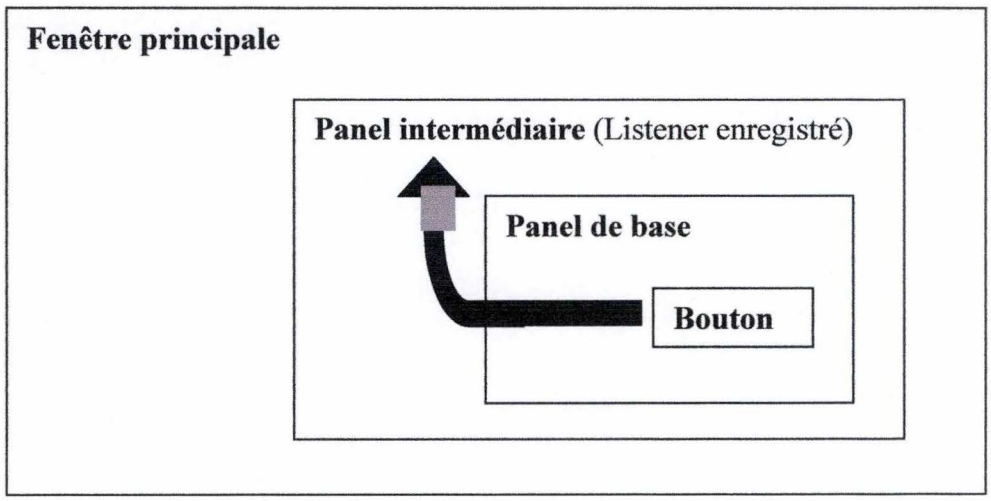


Figure 32 : Système événementiel AWT 1.1

Dans ce système, l'événement n'est donc pas propagé à travers une quelconque hiérarchie mais est dispatché vers les différents listeners enregistrés. Dans l'exemple, figure 32, seul le panel intermédiaire sera donc notifié de l'événement du bouton.

Les techniques et technologies utilisées ayant été expliquées, nous pouvons maintenant présenter la réalisation qui a été faite durant ce stage et montrer comment nous avons pu tirer parti de ces puissants outils.

4 Analyse et résultats

L'objectif de cette réalisation est de donner des bases solides à la plate-forme et non de l'implémenter dans son entièreté. La première étape consistera en l'intégration d'un package de corégistration, à savoir, AIR. Nous verrons comment ce dernier à pu être intégré à la plate-forme à l'aide de JNI. Nous découvrirons ensuite les premiers outils qui permettront d'établir une stratégie de corégistration automatique pour ce package. Enfin, cette partie sera clôturée par la présentation des différents outils de corégistration manuelle qui ont été implémentés.

La seconde partie sera axée sur l'aspect visualisation des images médicales. Pour cela, nous sommes reparti des concepts cités dans [M-Sey02] et de la première implémentation proposée dans [M-Vae03]. Beaucoup de choses ont néanmoins changé par rapport à ce que l'on trouve dans ce dernier. La couche 4 a été entièrement remodelisée et les couches 1 et 2 ont été partiellement revues dans la partie qui va suivre.

4.1 Partie 1 : Corégistration des images

Le premier et plus important point du travail porte sur la corégistration des images. Cette technique permet donc d'aligner deux images issues de modalités différentes et, le cas échéant, de pouvoir les fusionner. Pour mieux se situer par rapport aux travaux précédents, les ajouts et modifications vont s'effectuer au niveau de la couche 1, pour le stockage des données, et au niveau de la couche 2, pour l'intégration d'AIR à la plate-forme.

La corégistration est abordée sous deux aspect dans ce travail. Dans un premier temps, nous allons présenter la corégistration automatique voire semi-automatique de deux piles d'images. Cette corégistration est basée sur l'utilisation d'algorithmes extérieurs disponibles en opensource. Nous avons vu, en 2.2, qu'il existait plusieurs algorithmes pour corégistrer des images. Le but de cette première partie est de pouvoir intégrer le premier de ces algorithmes à la plate-forme, à savoir AIR. Nous allons y découvrir les problèmes relatifs à l'intégration de celui-ci ainsi qu'une solution de guidage à la corégistration La seconde partie abordera les outils de corégistration manuelle mis à la disposition du médecin dans la plate-forme. Ces outils permettront de faire un pré-alignement quand ceci est nécessaire, afin d'offrir aux algorithmes automatiques, un maximum de chances de succès.

Cependant, avant de pouvoir aborder ces deux parties de la corégistration, nous faisons une petite parenthèse afin de présenter un problème sous-jacent à ces deux techniques: il s'agit du problème de stockage des données.

4.1.1 Stockage des données:

La question du stockage des données brutes des images (Comment stocker les valeurs des pixels provenant des images médicales?) a du être revue car la solution prônée dans [M-Sey02] et adoptée, dans un premier temps dans [M-Vae03] pose quelques problèmes pratiques pour le reste de la réalisation.

[M-Sey02] propose de créer un ensemble de classes permettant de mémoriser les différents types de données. Comme nous l'avons découvert en 2.3, il est ici question de trois classes: ByteRawDataBuffer, ShortRawDataBuffer, UshortRawDataBuffer, toutes trois héritant de la classe abstraite RawDataBuffer. Après une longue réflexion sur le sujet, la méthode ne semble pas optimale. Nous devons jongler perpétuellement avec les types de données, adapter les algorithmes en conséquence, toujours faire attention au type de données manipulées. C'est difficilement envisageable. Même si l'idée de base ne semble pas mauvaise, elle nous attire beaucoup de soucis.

La solution retenue est de convertir toutes les données brutes en un seul et même type à savoir en *unsigned* short, dès le chargement de l'image médicale. Et, à la place de cet ensemble complexe de classes, se contenter d'un simple tableau unidimensionnel contenant les données. Le choix d'un tableau unidimensionnel peut paraître bizarre mais il est justifié lors de l'utilisation de JNI et des algorithmes de corégistration d'AIR. Il est en effet beaucoup plus aisé de faire passer des tableaux unidimensionnels de JAVA au C que des tableaux tridimensionnels.

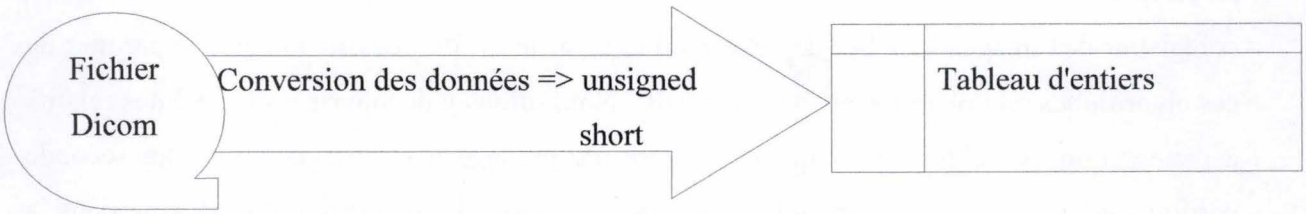


Figure 33 : Stockage des données

Le problème ayant été résolu, nous pouvons commencer la présentation de nos résultats en débutant avec la partie corégistration automatique.

4.1.2 La corégistration automatique des images

La corégistration automatique des images est effectuée à l'aide des algorithmes du package gratuit AIR présenté en 3.2.1. Ces algorithmes sont intégrés à la plate-forme à l'aide de la technologie JNI qui, elle aussi, fait l'objet d'une courte présentation en 3.2.2.

Une idée d'implémentation d'AIR est donnée dans [M-Vae03]. Elle propose une première approche à l'intégration d'AIR dans la plate-forme mais n'est pas approfondie dans ce mémoire. Sa méthode consiste, dans un premier temps, à convertir les fichiers DICOM vers le format de fichiers utilisé par AIR: Analyse. Une fois ces fichiers convertis, il faut les transmettre à AIR, qui effectue les opérations nécessaires à la corégistration et renvoie un fichier au format Analyse. Ces derniers doivent, de nouveau, être convertis avant de pouvoir être affichés dans la plate-forme de corégistration.

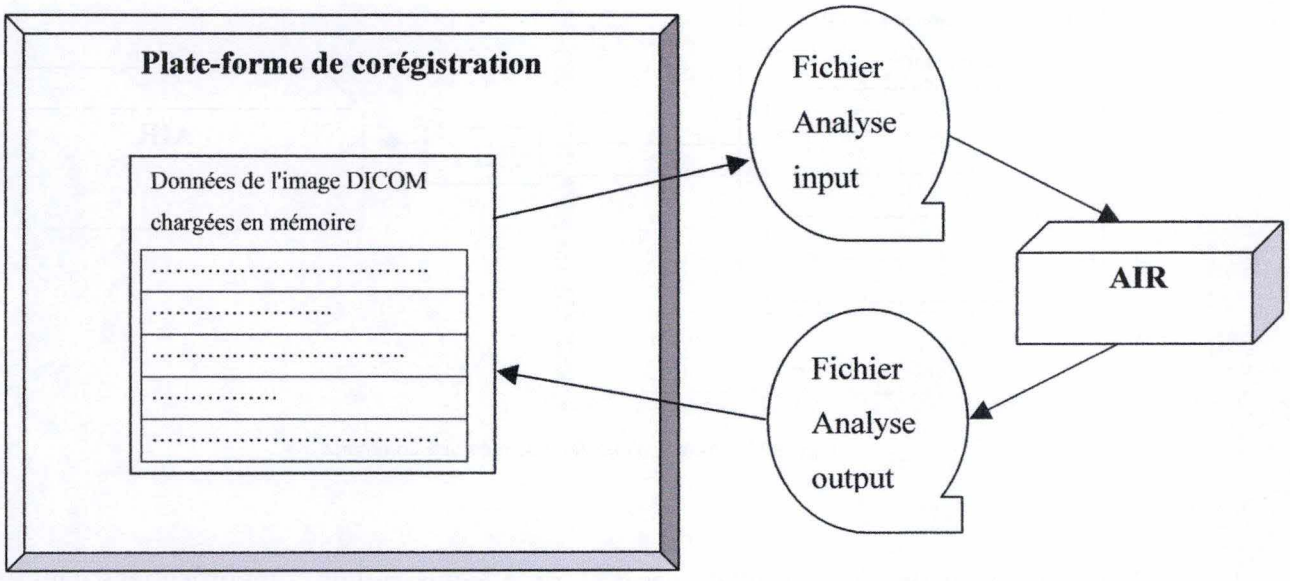


Figure 34 : Schéma représentant le fonctionnement d'AIR avec la plate-forme présenté dans [M-Vae03]

On peut facilement se rendre compte de la raison pour laquelle la méthode n'est pas viable. D'abord, les fichiers manipulés sont d'une assez grande taille. Cela varie entre quinze et soixante mégas par fichier. Les lectures et écritures multiples, que demande cette méthode font perdre un temps précieux. Ensuite, la duplication des données n'est pas non plus très adaptée, on se retrouve en effet avec deux fichiers Analyse temporaires et le fichier DICOM

de base, pour la même image. Des tests ont néanmoins été effectués pour tenter d'adapter cette méthode. La création d'un disque virtuel en mémoire centrale, par exemple. Mais ceci ne donne que des résultats peu probants. Cette méthode a donc été complètement abandonnée au profit de quelque chose de certainement plus difficile à mettre en œuvre mais d'aussi beaucoup plus propre et rapide dans son exécution, et plus en adéquation avec les objectifs d'intégration que l'on s'est fixés.

Le but est de pouvoir utiliser les possibilités étendues offertes par JNI afin de faire appel aux fonctions de corégistration écrites en C mais aussi, de faire communiquer dynamiquement ces fonctions avec les données et méthodes de JAVA. Il faut donc faire en sorte que l'algorithme d'AIR n'attende plus un fichier en entrée pour recevoir les données de l'image mais aille directement lire et écrire les données chargées en mémoire par JAVA à partir du fichier DICOM à traiter.

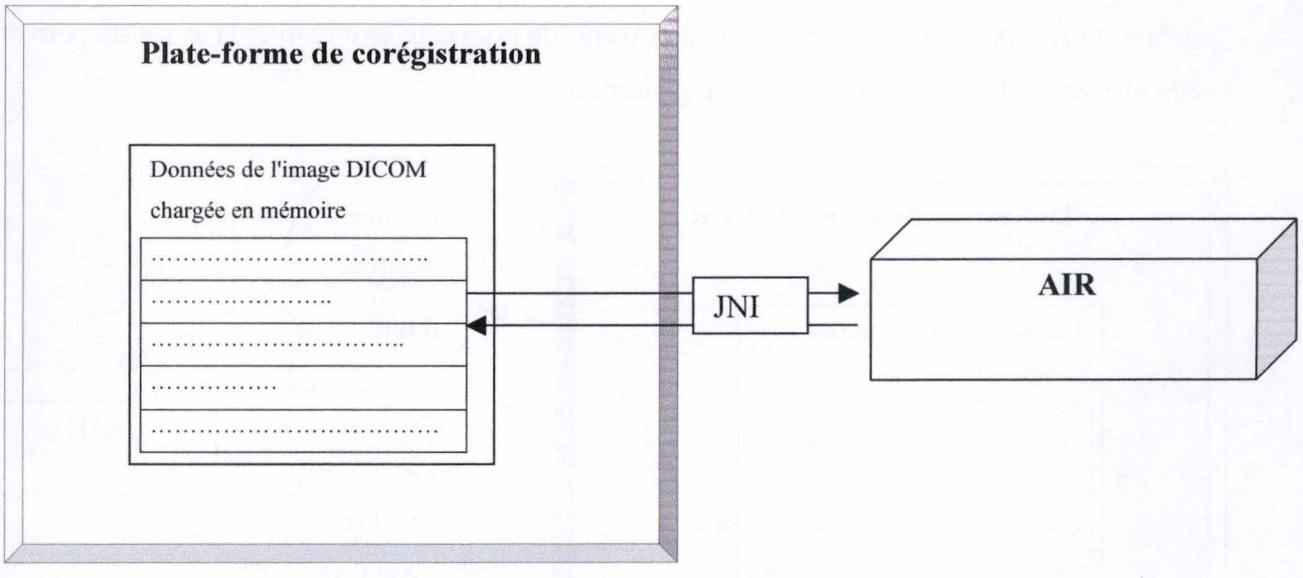


Figure 35 : Partage d'une zone de mémoire commune

Pour pouvoir arriver à ce résultat avec JNI, on effectue certaines modifications dans les fichiers source d'AIR. Il est impératif de bien cerner la structure de ce dernier avant de pouvoir commencer les adaptations. Cette structure peut se décomposer en trois grands niveaux et donne un diagramme présenté en page suivante.

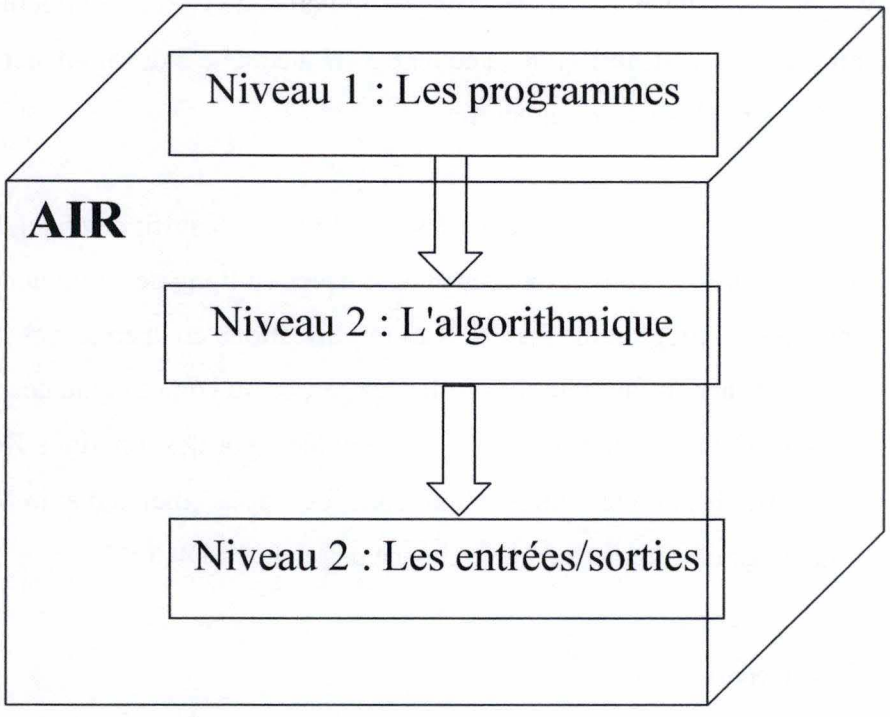


Figure 36 : Structure du package AIR

Cette découpe est tirée de l'expérience et du rétro-engeenering réalisé pour les besoins de l'application, il n'existe en effet que très peu d'informations quant à la structure et aux fonctions de bas niveau d'AIR.

Le niveau 1, que l'on nommera programme, reprend l'ensemble des fonctions "exécutables", ce sont elles qui contiennent les "main[12]". Nous avons vu, lors de la présentation d'AIR, que ce dernier est composé d'un certain nombre de programmes que l'on peut lancer en ligne de commande (resize, align_linear,...). Ce sont ces fonctions que l'on retrouve au niveau 1.

Au niveau 2 se trouvent les fonctions algorithmiques, ce sont ces fonctions qui contiennent les algorithmes mathématiques nécessaires à la corégistration automatique.

Enfin, le dernier niveau contient les fonctions nécessaires à la gestion des entrées/sorties. Ce sont donc les fonctions de lecture, de vérification et d'écriture des fichiers Analyse.

[12] Fonction principale en C, c'est par elle que le programme va débuter.

Pour pouvoir atteindre les buts fixés, et intégrer JNI au code C, il faut effectuer des modifications au niveau de la couche 1 et de la couche 3. La couche 2 devant rester intacte pour ne pas fausser les résultats de la corégistration.

Les modifications nécessaires à la couche 1 sont minimes. Il suffit de changer le nom des fonctions "main", car elles ne sont désormais plus utilisées en ligne de commande mais sont directement appelées du programme JAVA. Les modifications de la couche 3 sont, par contre, beaucoup plus subtiles. Il faut réussir à isoler les parties où l'on effectue des lectures/écritures vers les fichiers Analyse, afin de les remplacer par des fonctions JNI permettant d'aller lire/écrire les données directement dans les emplacements mémoire réservés par JAVA, pour le stockage des données en provenance des fichiers DICOM.

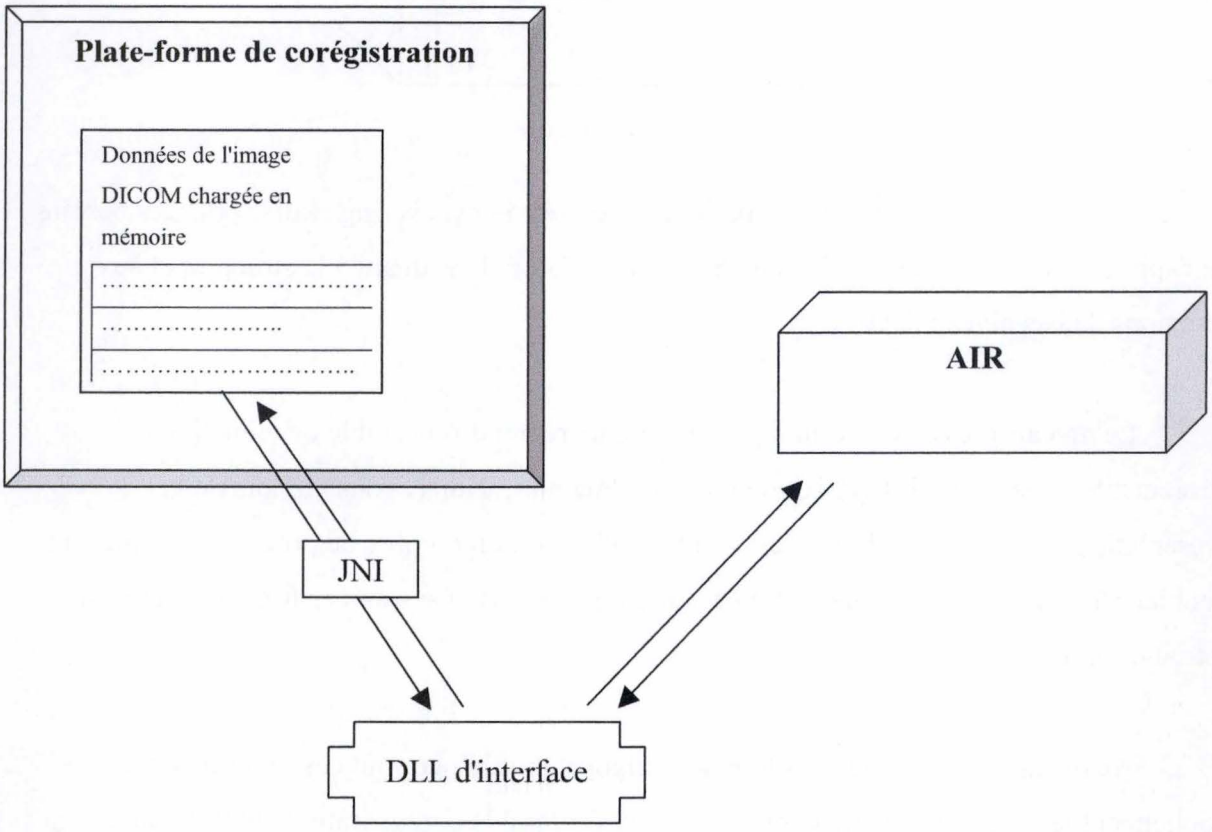


Figure 37 : Insertion d'AIR dans la plate-forme

Le composant "DLL d'interface" présent dans cette figure n'a pas encore été mentionné. Il s'agit d'une dll écrite dans le but d'interfacer AIR avec JNI. Les fonctions appelées à partir de JNI doivent avoir une syntaxe particulière. Nous décidons de ne pas toucher à la couche 1 (à part les noms des fonctions main) et d'utiliser des fonctions dans une

dll séparée ayant la bonne syntaxe. Ces fonctions servent donc de relais entre le programme JAVA (la plate-forme) et les programmes C (AIR). L'appel à la fonction "zoomer", citée lors de la présentation d'AIR au point 3.2.1, est illustrée en figure 38.

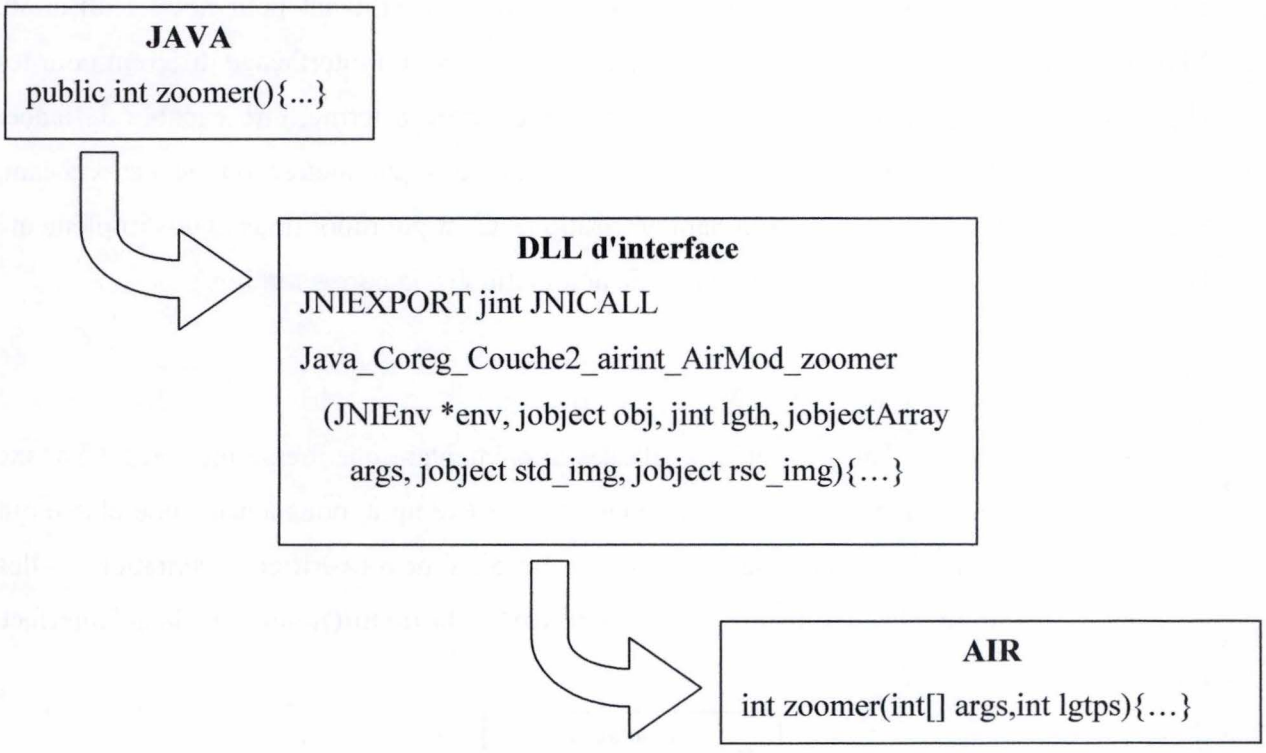


Figure 38 : Appel à la fonction "zoomer" d'AIR à partir de la plate-forme

4.1.2.1 Première ébauche d'aide à la corégistration

Jusqu'à maintenant, l'algorithme AIR a seulement été intégré à la plate-forme. Cela signifie que l'on peut appeler n'importe quel programme faisant partie du package, à partir d'un programme JAVA. Cela constitue déjà un grand pas en avant pour notre réalisation. Mais le but final de la plate-forme n'est pas juste de proposer un interfaçage différent pour les algorithmes de corégistration. Au delà de cela, elle devra, à terme, être capable de lancer automatiquement les programmes adéquats avec les bons paramètres ou, le cas échéant, proposer des outils d'orientation à la paramétrisation. C'est pourquoi nous avons implémenté l'outil suivant, qui constitue une première aide automatique à la corégistration.

Principe:

Nous disposons d'un ensemble de classes implémentant une même interface. Chacune de ces classes est relative à un problème inhérent. Par exemple, nous aurons une classe qui sera chargée de contrôler la taille des images, une autre qui devra vérifier l'orientation,...elles devront obligatoirement redéfinir les méthodes **testit()** et **launchit(),** définies dans l'interface implémentée.

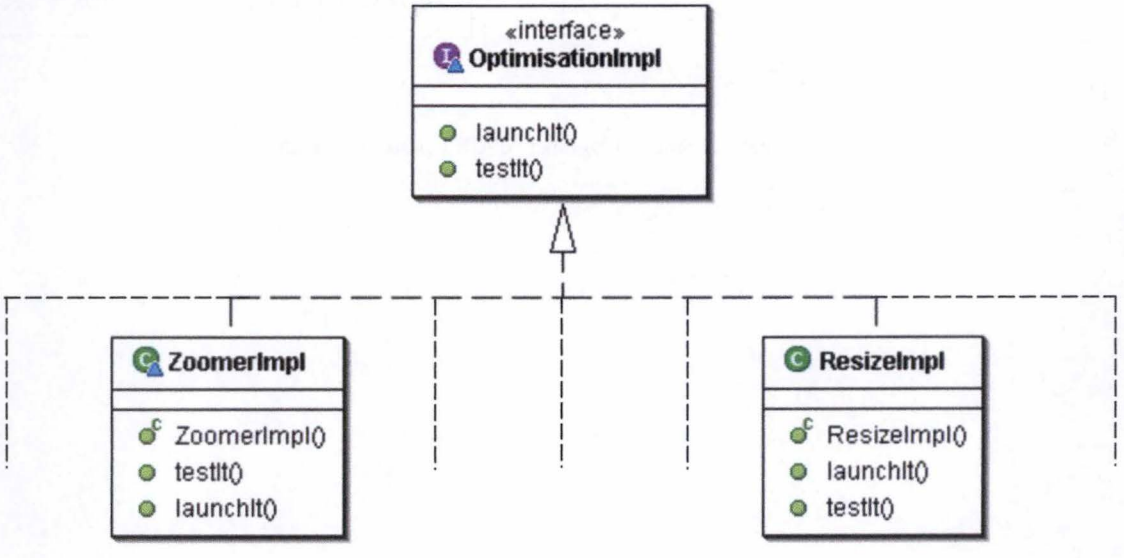


Figure 39 : Classes d'aide à la corégistration implémentant une même interface

La première de ces méthodes sert à lancer une phase d'analyse relative au problème traité par la classe. Pour donner un exemple simple, on pourrait avoir une classe chargée de

comparer la taille des deux images à aligner. Si la taille des images est différente, on effectue un marquage et on sauvegarde les caractéristiques (en l'occurrence la taille des images).

Une fois cette phase d'analyse terminée, le lancement de la deuxième méthode (**launchit()**) sera définie par le marquage qui aura été fait pendant l'analyse. Si, par exemple, l'analyse de la taille des images révèle une disparité entre ces dernières, la méthode **launchit()** sera appelée et permettra de lancer le programme AIR, ou les paramètres à passer permettant de corriger cela.

Toutes ces classes d'aide à la corégistration sont gérées à l'aide du mécanisme de réflexivité présenté en .... Dès le moment où elles implémentent l'interface adéquate et, donc, redéfinissent les méthodes **testit()** et **lanchit()**, elles pourront être intégrées, d'une manière totalement automatique et transparente, à la plate-forme. Jusqu'à maintenant, deux outils sont implémentés. Le but n'étant pas, dans ce mémoire, de les répertorier tous, mais bien de donner une ligne de conduite que les prochains pourront suivre afin de mettre sur pied les techniques nécessaires.

Point de vue utilisateur:

La fenêtre de corégistration se présente comme suit :

Figure 40 : Interface utilisateur de la partie corégistration automatique

On observe deux panels contenant chacun une image d'un même cerveau provenant de modalités différentes ; la première provient d'une résonance magnétique tandis que la seconde provient d'un PET Scan. Le but est de faire aligner ces deux images par AIR

Sur la gauche de la figure 40, la partie des outils consacrés à la corégistration automatique. La partie supérieure gauche, qui est entourée en bleu. Cette partie d'interface sert à faire appel aux programmes extérieurs de corégistration (dans notre cas particulier, AIR). Les deux premiers boutons servent à lancer l'aide à la corégistration qui vient d'être présentée. Le bouton "Analyser" sert à lancer les méthodes **testit()** de toutes les classes d'aide à la corégistration disponibles. A chaque test positif, une ligne avec le nom du programme qu'il sera nécessaire de lancer vient s'ajouter dans la liste d'actions à effectuer; s'il est négatif,

rien ne sera ajouté. Dans l'exemple de la figure 40, on voit qu'un redimenssionnement de l'image et qu'un traitement sur les voxels seront nécessaires.

La phase d'analyse terminée, le bouton "Exécuter" passe au vert et permet alors de lancer les méthodes **launchit()** de chaque classe d'aide à la corégistration. Cette méthode se chargera, dans notre cas, de faire appel au programmes AIR nécessaire à l'alignement des images.

4.1.3 La corégistration manuelle

Comme nous l'avons déjà fait remarquer à plusieurs reprises, la corégistration automatique n'est pas toujours garantie de résultats. Si les images ont de trop grandes disparités, il devient difficile, pour l'algorithme, de mener sa tâche à bien. C'est pourquoi des outils sont mis en place afin que l'utilisateur puisse réaliser un pré-corégistration manuelle quand la corégistration automatique ne donne pas de bons résultats.

Le premier outil disponible sert à renverser la pile d'images selon un axe choisi.

Principe:

Dans certains cas, les piles d'images ne sont pas obtenues dans le même sens avec les différents appareils. Le programme permet de faire une inversion sur les trois axes (X, Y, Z), simplement en recopiant le buffer des données brutes dans des sens différents. La figure 41 montre un exemple simple d'inversion sur l'axe Y.

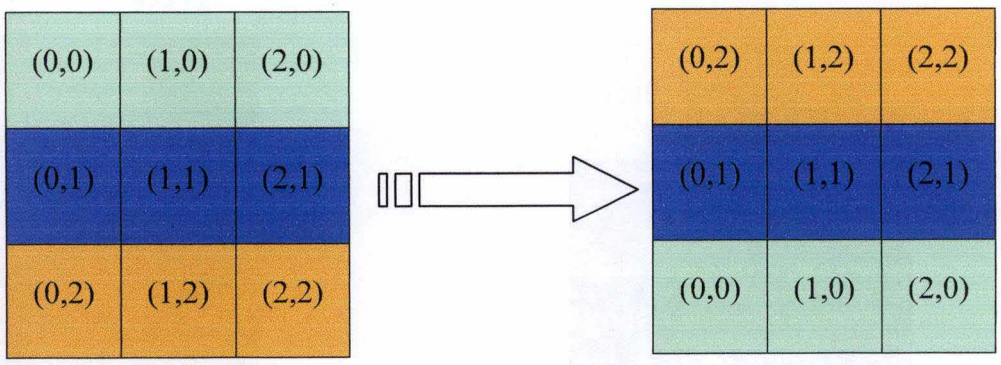


Figure 41 : Exemple d'inversion d'une image 2D sur l'axe Y

Point de vue utilisateur:

Voici la fenêtre permettant à l'utilisateur d'effectuer les inversions suivant l'axe choisi. Les boutons en rouges sont les actions que l'utilisateur choisi d'exécuter.

Figure 42 : Inversion d'une pile d'images suivant un axe particulier

Le second outil disponible est l'outil « Threshold ». Il permet de définir un seuil de visibilité des pixels.

Principe:

On spécifie deux valeurs, la minimale et la maximale. Les pixels, qui ont une valeur plus petite que la minimale ou plus grande que la maximale, sont mis à zéro et ne sont donc plus visibles.

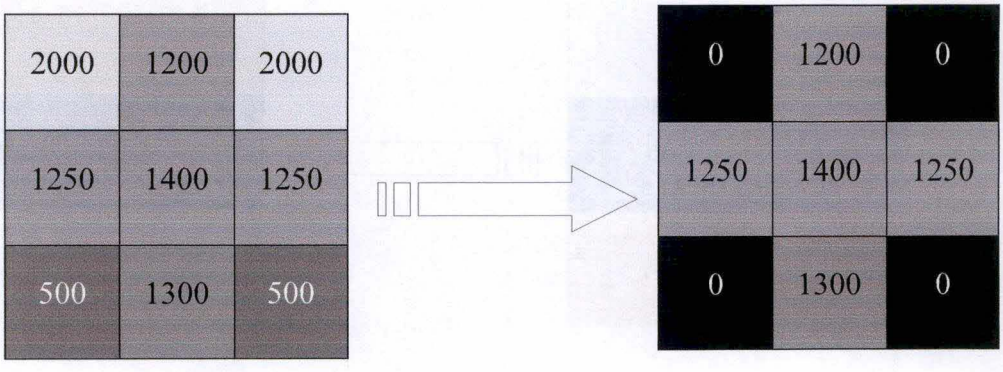


Figure 43 : Exemple de seuillage avec une valeur minimum à 600 et une valeur maximum à 1800

Cet outil est intéressant lorsque les images proviennent de la résonance magnétique par exemple, afin de faire disparaître les zones non désirées comme « la cage osseuse » qui entoure le cerveau et qui est affichée en blanc. Comme elle possède une forte intensité lumineuse, elle sera plus vite éliminée avec le threshold que le reste de l'image. Il faut cependant se servir de cet outil avec parcimonie car certaines parties de l'image, utiles elles, peuvent aussi disparaître…

Point de vue utilisateur:

La fenêtre du threshold se présente comme suit :

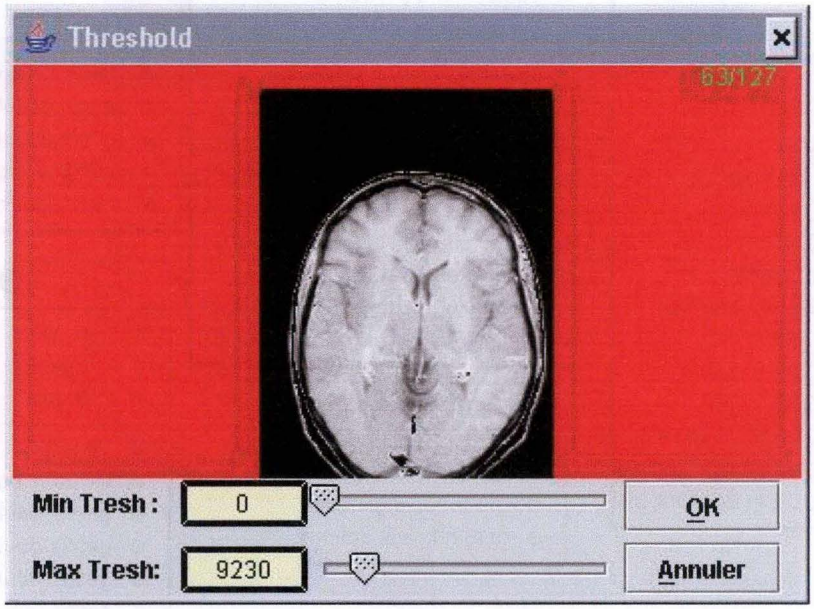


Figure 44 : Fenêtre de réglage du threshold

L'outil threshold n'est pas un outil efficace à 100% mais est toujours mieux que rien. Pour pouvoir avoir un résultat plus probant, un atelier de découpe pourrait être ajouté au programme avec des algorithmes de détection des contours et d'ajustements manuels.

Le troisième outil disponible pour la corégistration manuelle est appelé "outil d'alignement manuel". Cet outil permet d'effectuer des translations/rotations sur le buffer (la pile d'images) afin de l'aligner manuellement avec l'autre. Attention, il ne faut pas confondre ces translations/rotations avec les outils de visualisation qui seront présentés en 4.3. Il y a une différence fondamentale ! Les outils de visualisation n'altèrent en aucun cas le buffer des

données brutes; il le présente simplement sous différentes formes. Les outils de corégistration, eux, manipulent directement ce buffer et les changements sont opérés définitivement.

Principe:

Ces opérations sont effectuées grâce à un ensemble de classes implémentant les différentes opérations matricielles de base qu'il est possible de réaliser dans l'espace.

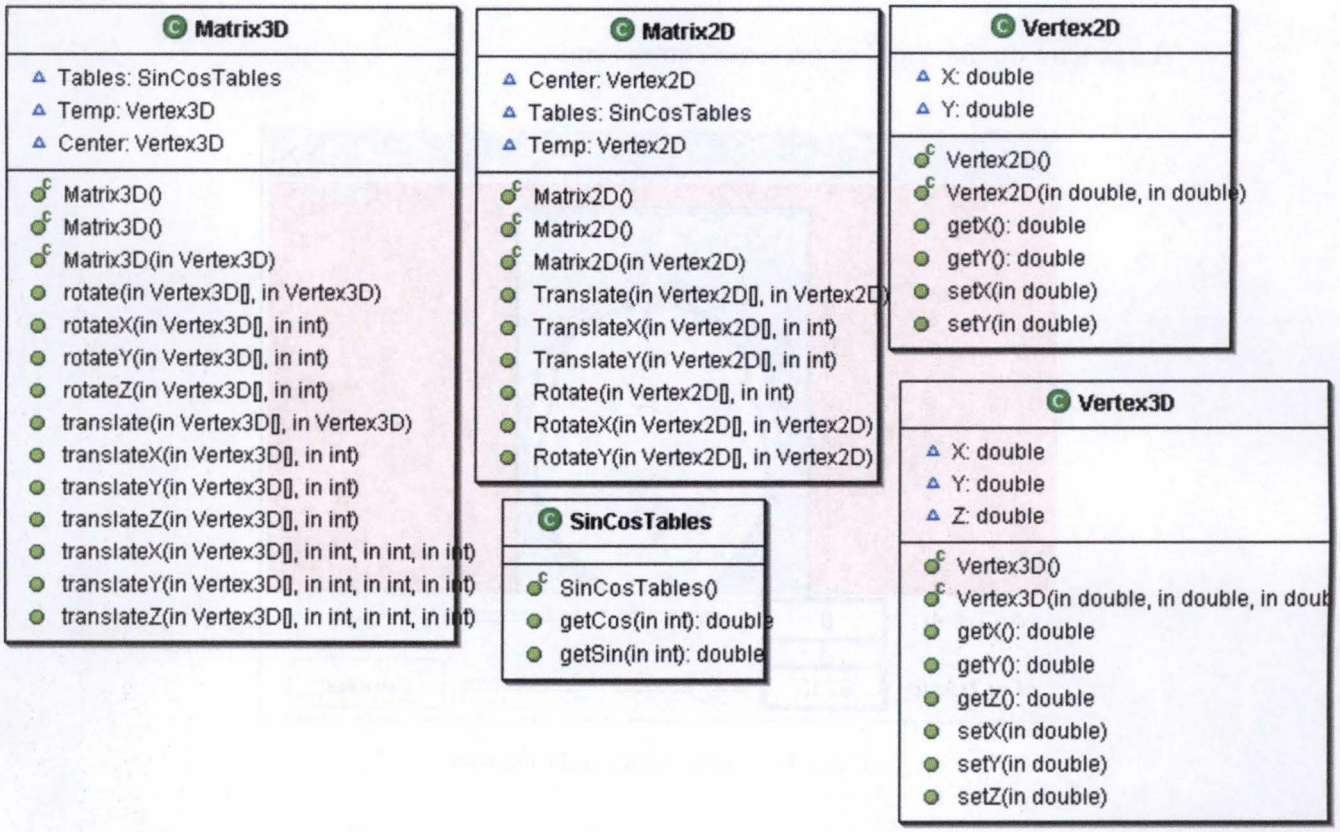


Figure 45 : Classes créées pour les opérations matricielles

Les classes Vertex2D et Vertex3D servent respectivement à stocker les coordonnées d'un pixel dans le plan ou dans l'espace. Les classes Matrix2D et Matrix3D servent quant à elles, à effectuer les opérations matricielles de base pour le plan et pour l'espace, à savoir: la translation, la rotation d'un Vertex. Enfin, la classe SinCosTables est présente ici pour des raisons de performance. On a besoin des cosinus et sinus des angles pour pouvoir effectuer des rotations. Comme le pas de rotation minimum est fixé à 1 degré, plutôt que de recalculer

les valeurs des cosinus et sinus pour chaque rotation, on précalcule celles-ci dans des tables de 360 entrées, reprenant chaque valeur de sinus/cosinus pour chaque angle. Ceci nous permet de faire un gain assez conséquent de performances dans l'application.

Point de vue utilisateur:

Voici, en figure 46 l'écran d'alignement manuel. Il dispose donc d'une interface permettant de réaliser des translations et rotations de l'image. Suivant la vue sélectionnée (axiale, sagittale ou coronale), les translations/rotations s'effectuent suivant un certain axe. Par exemple, la vue axiale permet de faire des translations suivant les axes X et Y tandis que la vue sagittale permet d'effectuer ces même translations sur l'axe des Y et des Z.. Enfin, une grille peut être affichée afin d'aider l'utilisateur à se situer.

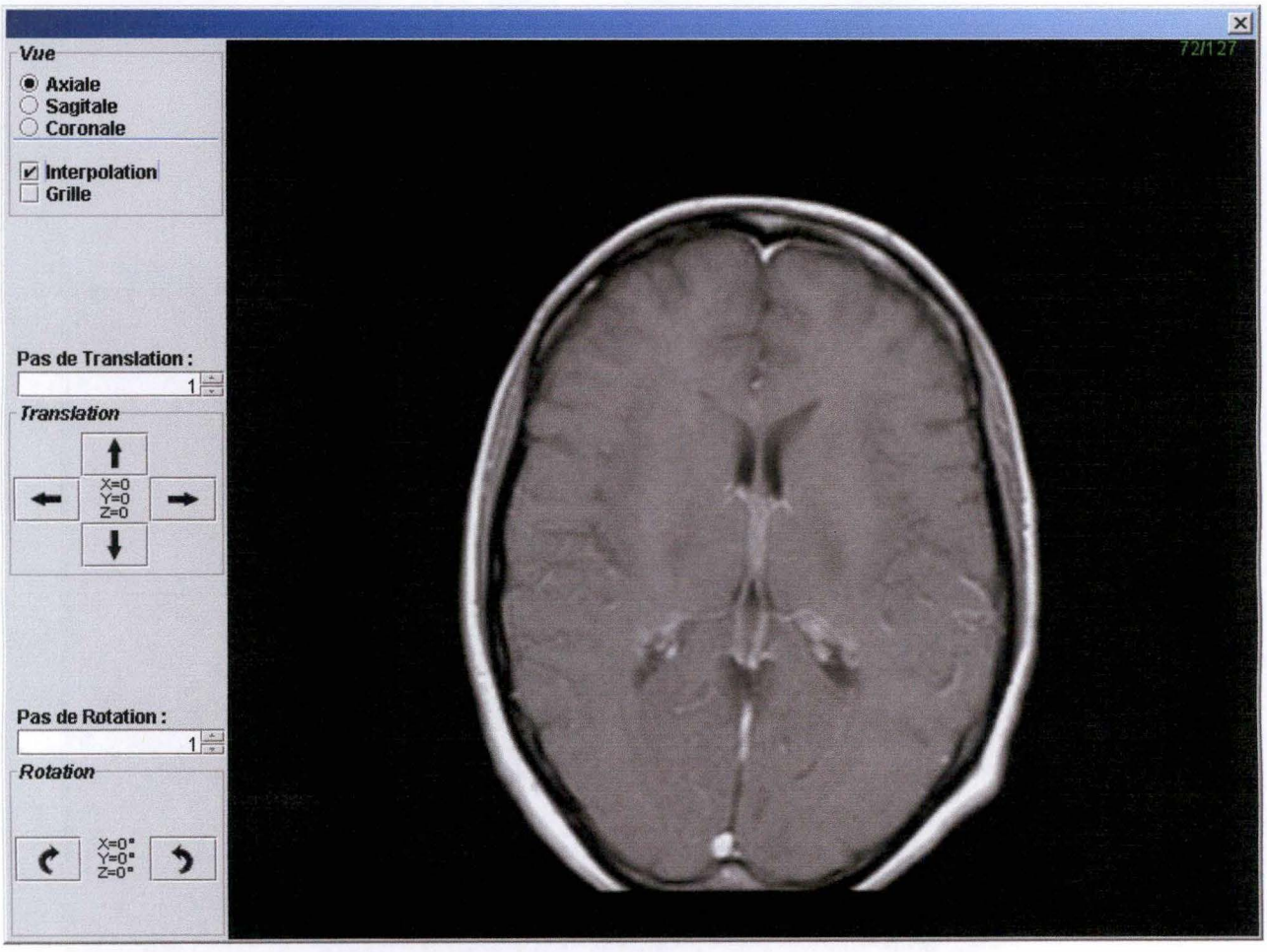


Figure 46 : Outil d'alignement manuel

Dernier outil de corégistration manuelle implémenté, l'outil de centrage/alignement des images semi-automatique. Ce dernier sert à mettre les deux piles d'images en même position sur l'écran.

Principe:

Lorsque l'on choisit d'utiliser cet outil, une fenêtre est ouverte avec une des deux piles d'images. Le curseur de la souris est alors remplacé par une croix prenant tout l'écran. Cette croix sert à pointer le centre approximatif de l'image. Une fois cette opération réalisée, la pile d'images est remplacée par l'autre pile (celle à aligner) et on réitère les mêmes opérations. Un fois ces opérations terminées, le programme calcule les déplacements entre les différents points marqués et aligne les images. Une liste sur la gauche renseigne les actions à effectuer et maintient aussi l'historique des actions effectuées.

Point de vue utilisateur:

La fenêtre de centrage/alignement se présente comme suit:

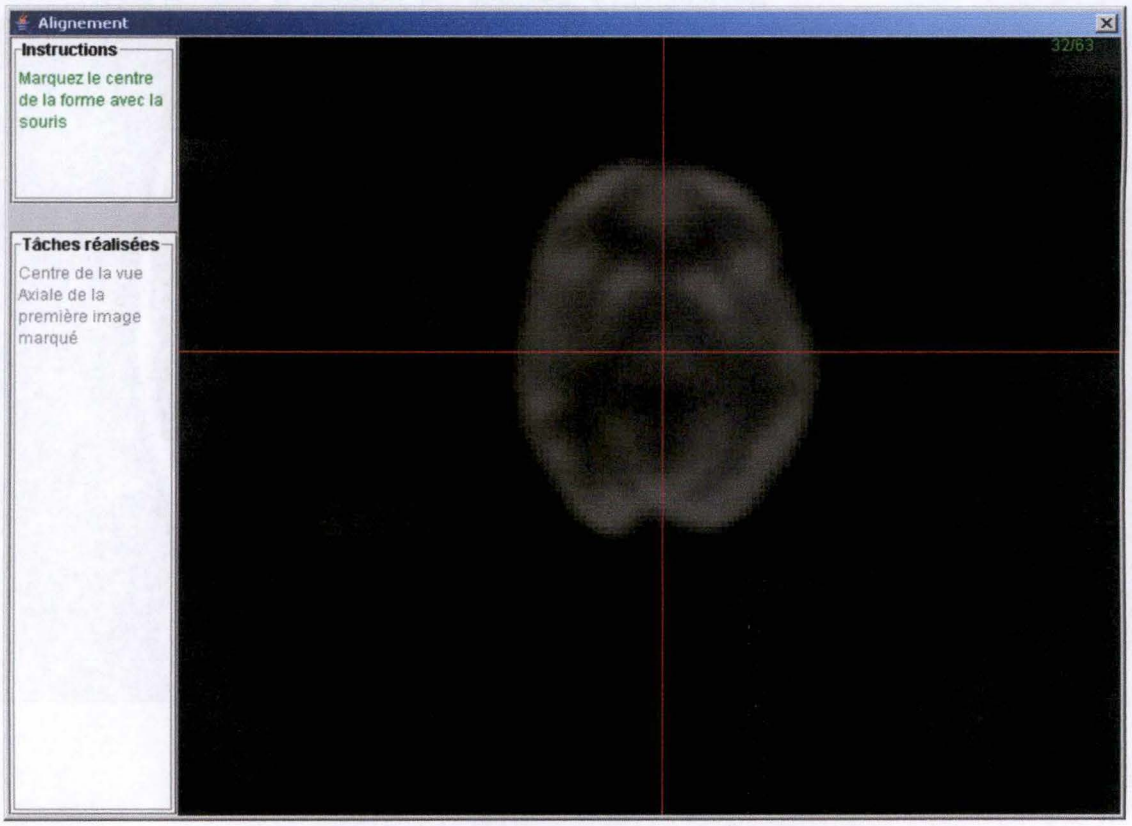


Figure 47 : Outil de centrage/alignement semi-automatique

Il est évident qu'après la corégistration manuelle, une corégistration automatique peut être lancée. Celle-ci aura d'autant plus de chances d'aboutir que la corégistration manuelle a été correctement réalisée.

4.3 Partie 2 de l'application : Visualisation des images

Dans cette partie, une étude a été faite sur les différents moyens et méthodes possibles de visualisation des images médicales et, plus particulièrement, des images médicales corégistrées. Les problèmes étaient multiples et pas toujours évidents à contourner.

4.3.1 Génération des images

Il a fallu trouver un moyen pour consommer le moins de mémoire possible et être assez rapide dans la visualisation des images. En effet, comme nous l'avons déjà présenté précédemment, nous nous trouvons ici face à des piles d'images (parfois jusqu'à 128 images et peut-être plus dans le futur). Ces images doivent pouvoir être parcourues en temps réel et sans souffrir d'un quelconque ralentissement.

Pour la visualisation des images, la représentation d'un pixel à l'écran prend une forme particulière genre ARGB, RGB,CMYK[13],... or, on ne dispose que de données brutes de type unsigned short .

Plusieurs essais ont été faits afin de choisir le meilleur compromis. La première approche consiste à générer une pile d'images dans un format compréhensible par java à partir des données brutes. Cette méthode permet de ne souffrir d'aucun défaut de rapidité mais s'avère totalement inutilisable pour les raisons suivantes :

- La consommation de mémoire devient trop importante. Les images peuvent avoir des tailles de 256*192 pixels, si l'on a une pile de 128 de ces images avec chaque valeur de pixel stockée sur un entier, on arrive à une demande mémoire beaucoup trop importante (256*192*128*4 bytes) pour une pile d'images. Or, le programme doit pouvoir être capable de manipuler trois piles de ce genre à la fois (une pour l'image maître, une autre pour l'image esclave, et une dernière pour la fusion des deux premières).

[13] Ces modes donnent la façon dont un pixel est codé, soit par intensité des couleurs primaires Rouge, Verte et Bleue, soit par la méthode CMYK (cyan, magenta et jaune),...

- La cohérence entre les données brutes, provenant des fichiers Dicom, et la pile d'images devient beaucoup trop compliquée à maintenir. Avec cette technique, on a un duplicata visuel (la pile d'images) qui doit respecter le buffer de base (les données brutes) quelles que soient les modifications que l'on peut faire sur ce dernier. Donc, à chaque modification des données brutes (avec les outils de 4.1 par exemple), il faut de nouveau régénérer la pile d'images…

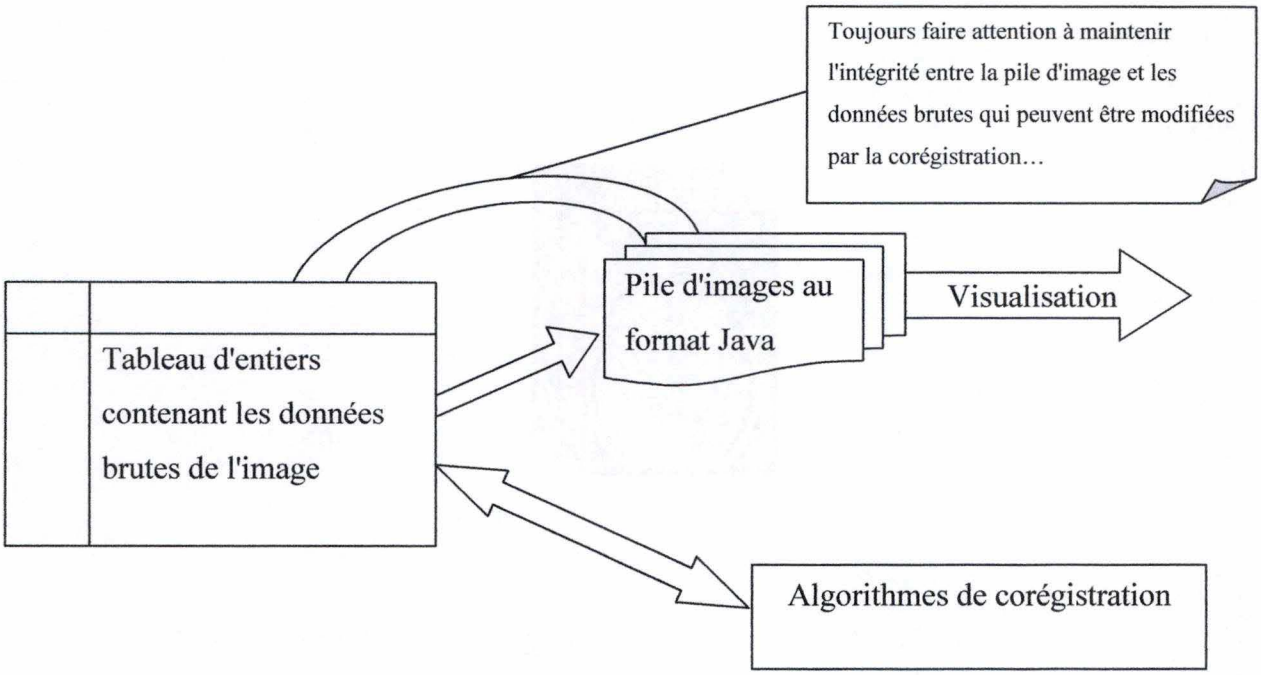


Figure 47 : Première solution envisagée : Générer une pile d'images à partir des données brutes

Finalement, la solution choisie est de générer les images « au vol » donc, en calculant une image 'java ' en temps réel, lorsque l'on en a besoin, à partir des données brutes. Avec quelques optimisations, cette technique se révèle être la plus adéquate (rapidité acceptable et consommation mémoire diminuée de moitié, de plus la cohérence est toujours assurée, quels que soient les changements effectués sur les données brutes).

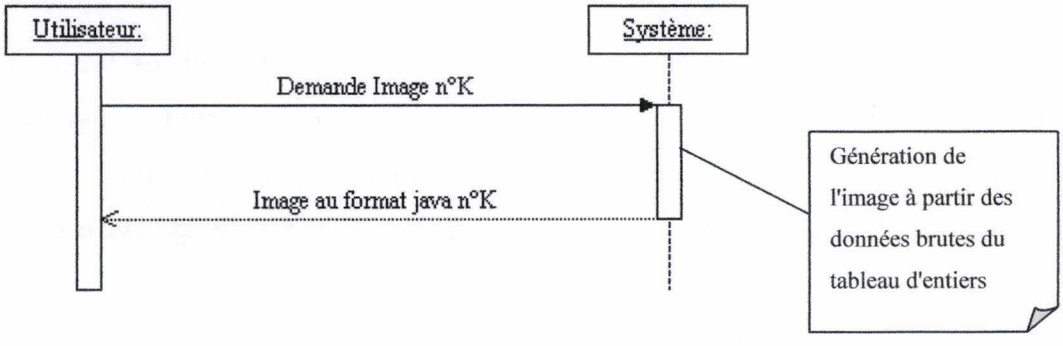


Figure 48 : Solution adoptée : Génération des images au vol

4.3.2 Présentation des images

Principe:

Comme nous l'avons déjà remarqué, les images médicales peuvent être présentées suivant différents point de vue car les modalités produisent généralement des piles d'images représentant les différentes coupes d'un organe.

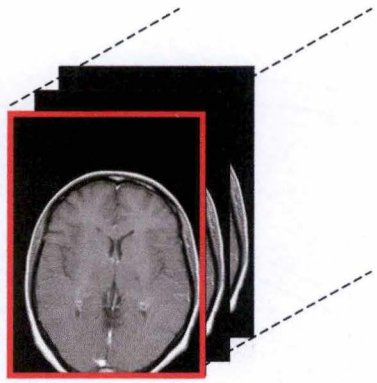


Figure 49 : Pile d'image provenant d'une IRM

Si l'on prend chacune de ces images et qu'on les arrange les unes derrière les autres, nous obtenons un volume présenté en figure 50.

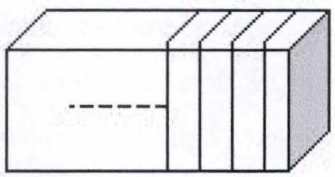


Figure 50 : Volume formé par l'arrangement des images

On peut alors remarquer qu'il est possible, non seulement, de lire et d'afficher une image dans sa forme originale mais aussi que l'on peut lire à travers le volume afin de pouvoir reconstruire des images 2D selon un plan déterminé, dans le volume. Les vues les plus courramment obtenues sont les vues axiale, sagittale et coronale, qui correspondent aux trois axes de l'espace. [P-Naz02].

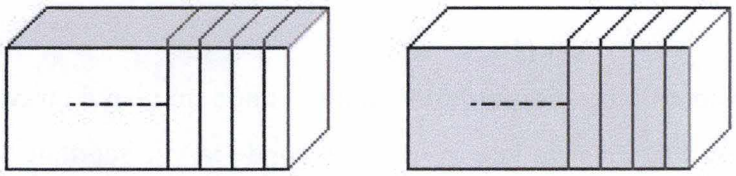


Figure 51 : Lecture longitudinale d'une pile d'image afin d'avoir les vues Sagittales et coronales

Ces vues sont, généralement, celles que l'on retrouve dans la plupart des logiciels de visualisation d'images médicales. Néanmoins, d'autres possibilités existent. Plutôt que d'avoir des coupes axiales, sagittales et coronales, on peut choisir de trancher le volume d'une façon différente, en spécifiant un angle d'attaque. Les images 2D sont donc générées selon cet angle.

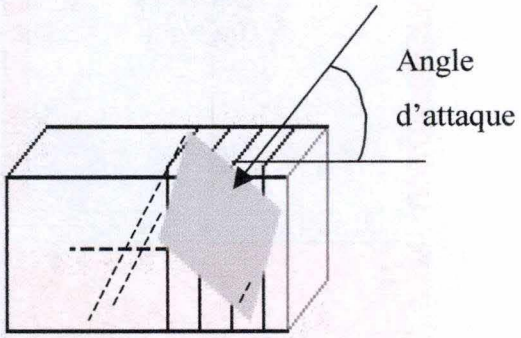


Figure 52 : Affichage des images selon un angle d'attaque

Pour pouvoir effectuer ceci, les même classes que pour la corégistration manuelle sont utilisées[14].

Attention, il ne s'agit ici que d'une technique de visualisation, **le buffer des données brutes reste intact**. Contrairement à la corégistration, on ne change rien dans ce dernier, et les classes représentant les opérations matricielles ne sont plus utilisées pour modifier le buffer des données brutes mais, pour le lire d'une façon différente.

[14] Se reporter à la figure 29.

Point de vue utilisateur:

Dans le panneau "Visualisation", on a à disposition un plan de travail divisé en quatre fenêtres. La première contient la vue axiale, la seconde la vue sagittale, la troisième la vue coronale et enfin, la dernière sert à la visualisation libre.

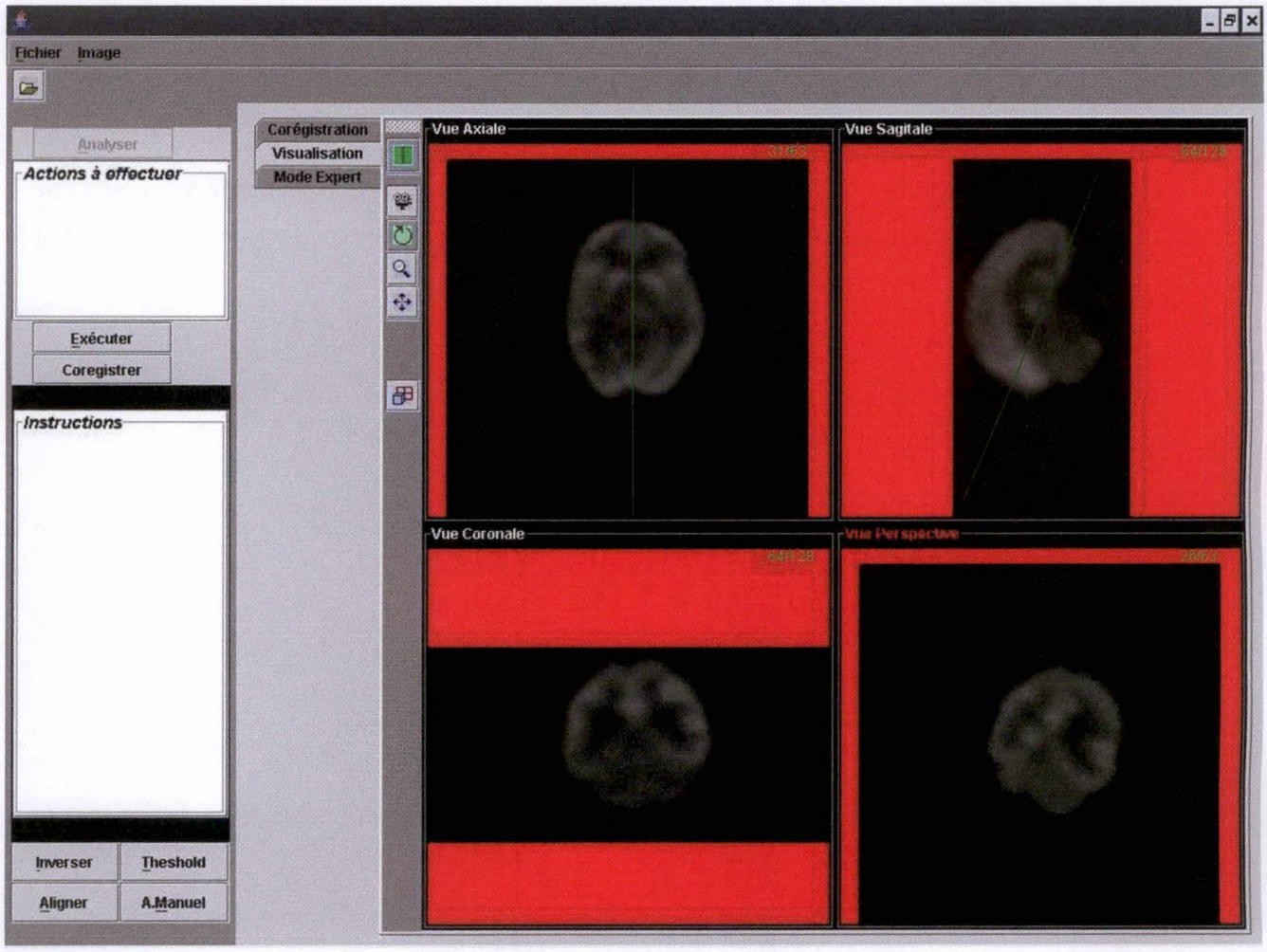


Figure 53 : Présentation des différentes vues disponibles

Dans la plate-forme, la vue 3D fonctionne suivant le principe suivant: il est possible de faire apparaître une ligne verticale sur les trois vues de base. Si l'on sélectionne le mode approprié, nous pouvons faire tourner la ligne sur son axe. La vue 3D subit alors une rotation équivalente. Par exemple, si l'on fait tourner la ligne verte de 45° dans la vue sagittale, la vue 3D subira une rotation de 45° sur l'axe des X. Il est évidemment toujours possible de naviguer dans les différentes coupes, dans la vue 3D.

4.3.3 Outils de visualisation

Les différents outils de visualisation vont maintenant être présentés. Ces outils sont, entre autres, représentés par les petites icônes sur la gauche du panel de visualisation. Pour plus de clarté, l'icône correspondant à l'outil (l'icône en vert représentant l'outil lorsqu'il est activé) sera reprise à chaque fois et une brève description de celui-ci sera alors donné.

 	La petite caméra représente l'outil de navigation dans les tranches de la pile d'image. On peut ainsi se promener dans le volume d'image en effectuant un « drag » à la souris sur la vue souhaitée.

 	La flèche de rotation représente l'outil par lequel l'angle d'attaque de la vue perspective est choisi. On choisit tout d'abord une vue « 2D », une ligne sera tracée dans cette vue (la ligne verte sur la figure 53) cette ligne représente la rotation que va subir l'image dans la vue perspective suivant un certain axe (l'axe étant déterminé par la vue dans laquelle on fait tourner la ligne. Par exemple, la rotation de la ligne dans la vue axiale permettra de faire subir une rotation sur l'axe des Y à l'image de la vue perspective…).

 	La loupe représente, comme l'on s'en doute, l'outil de zoom. Il est possible de zoomer/dézoomer sur n'importe quelle image de n'importe quelle vue. Afin de ne pas altérer la qualité des images, une interpolation bicubique est réalisée sur les images zoomées. Cette interpolation a lieu lorsque le drag de la souris est terminé. Tous les outils présentés ici fonctionnent en effet suivant le principe du drag&drop. Lorsque par exemple on veut se balader dans les différentes tranches de la pile d'images, que l'on veut effectuer un zoom ou une translation sur une image, il suffit de sélectionner l'outil désiré dans la barre d'outils sur la gauche et d'effectuer un drag dans une des vues. Lorsque ce drag est effectué, les artifices du genre interpolation sont désactivés afin de maintenir une vitesse d'affichage acceptable. Ce n'est que lorsque le « drop » est réalisé que l'interpolation est réactivée.

 	Les flèches croisées permettent d'effectuer une translation de l'image (PAN), afin, si besoin est, de la recentrer par exemple…

	Enfin, la dernière icône sert à remettre les images dans leur état initial. Si l'on a effectué des zooms, translations,… il suffit de cliquer sur cette icône pour que tout soit recentré, remis à sa place, à la bonne dimension.

Toutes ces icônes sont visibles sur la figure 54.

Il est également possible d'accéder à ces outils (et aussi quelques autres) par un menu contextuel accessible par le clique droit de la souris :

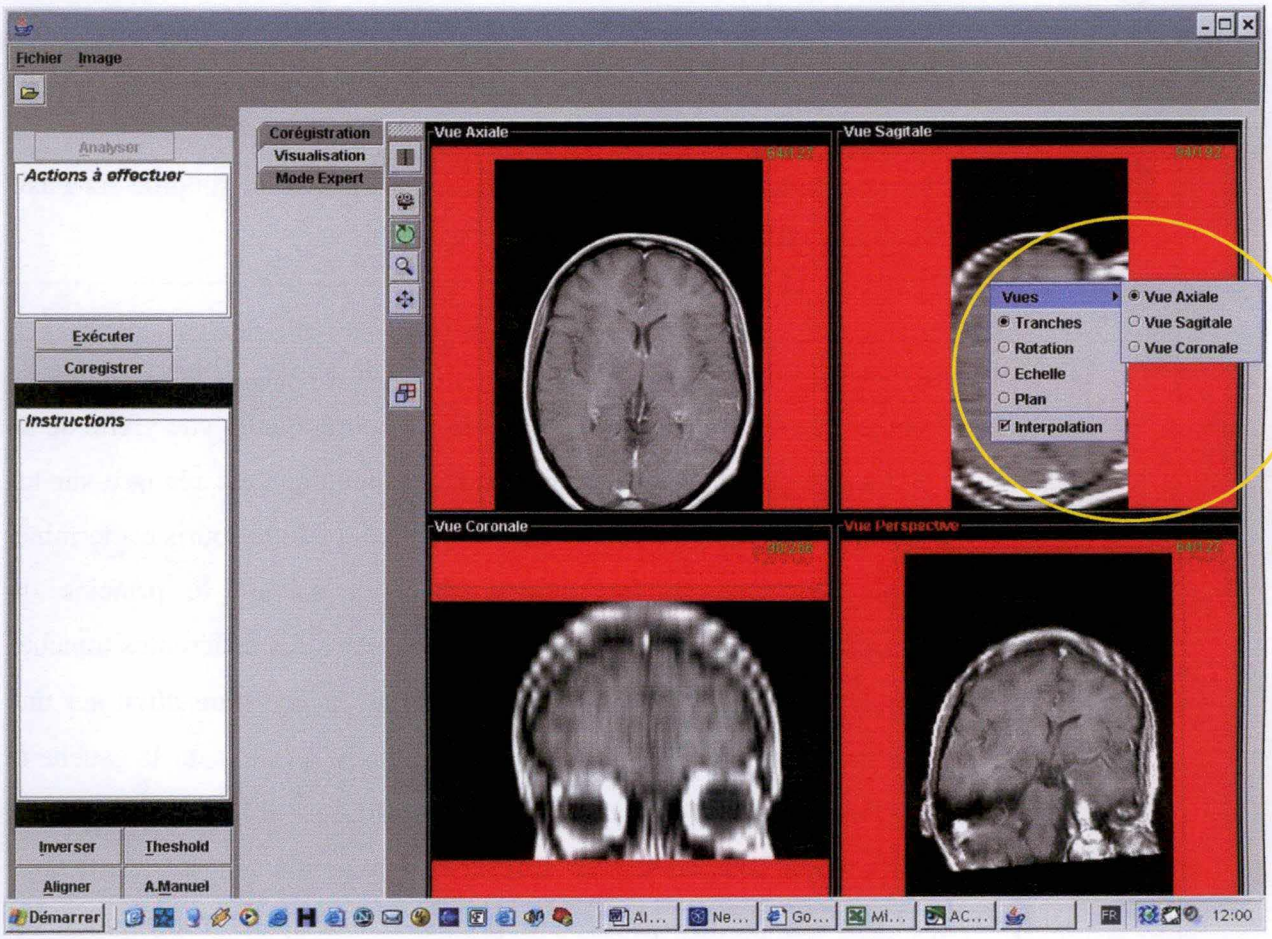


Figure 54 : Menu contextuel

Ainsi, le premier menu permet de changer la vue de n'importe quel panel de visualisation (sagittale, coronale ou axiale). Les quatre points suivants reprennent les mêmes options que la barre d'outils de gauche mais individuellement pour chaque panel (les options du menu contextuel ne sont valables que pour la vue sur laquelle on a cliqué). La dernière option permet d'activer/désactiver l'interpolation. L'interpolation permet d'obtenir des images exemptes de carrés lors du grossissement de celles-ci.

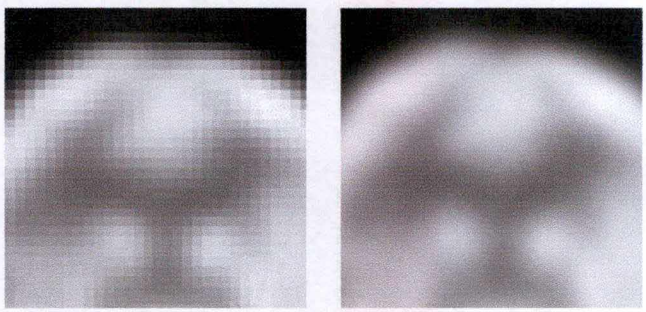


Figure 55 : Image sans interpolation VS image avec interpolation

Le principe est d'inventer des pixels, là où il n'y en a pas (lors d'un grossissement) en effectuant une moyenne des intensités des points adjacents à la zone à inventer. Elle demande généralement beaucoup de ressources, c'est pourquoi une petite astuce a été utilisée, pour ne pas que l'utilisateur ressente de ralentissements à l'utilisation. L'interpolation est en effet désactivée lors de l'utilisation des outils de visualisation. Par exemple, lors d'un zoom, l'utilisateur clique sur l'image et effectue un drag à la souris qui permet de spécifier le facteur de zoom. L'image est alors agrandie en temps réel sur l'écran de l'utilisateur. Pendant le drag, l'interpolation est désactivée afin de ne pas avoir de saccade. Elle sera automatiquement réactivée lorsque l'utilisateur lâchera le bouton de la souris, marquant, par la même occasion, la fin du zoom. Cette technique est utilisée pour tous les outils présentés au point 4.3.3.

On pourrait également classer le mode « expert » dans la catégorie "outils de visualisation". Ce dernier reprend un panel unique mais beaucoup plus gros que les quatre premiers afin de pouvoir afficher des images à leur taille réelle ou en beaucoup plus gros si besoin est…

Figure 56 : Panneau de visualisation Expert

Le même menu contextuel que les autres est accessible sur ce panel[15].

Pour terminer ce tour des outils servant à la visualisation d'images, deux outils très importants en imagerie médicale sont également implémentés.

Le premier est un outil permettant de jouer sur la luminosité et le contraste de l'image.

[15] Comme sur tous les panels de visualisation de l'application

Principe:

La luminance d'une image se définit comme étant *la moyenne des valeurs de niveaux de gris de chaque pixel qui la compose*. Si l'on considère la représentation monochrome d'une image, on peut définir 256 niveaux de luminance différents. [M-Zuy03].

Le réglage de l'intensité lumineuse d'un pixel est donc relativement simple à réaliser. Il suffit de procéder par incréments sur l'intensité de la palette utilisée par l'image. L'intensité générale des couleurs de l'image est donc augmentée comme illustré en figure 57.

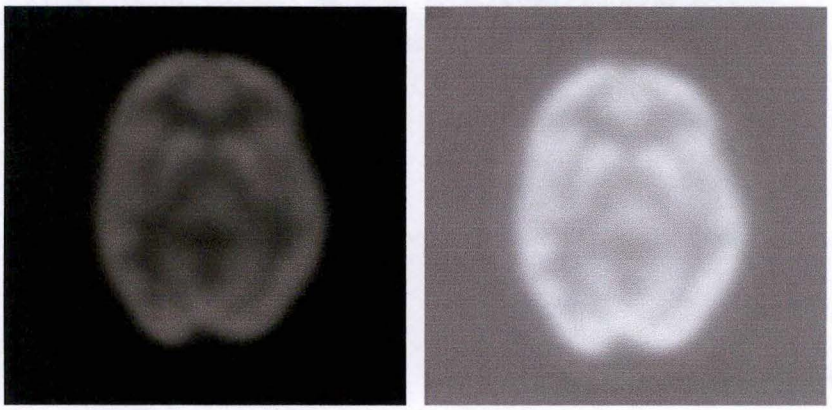


Figure 57 : Modification de la luminosité

Le contraste, lui, se définit comme l'opposition marquée entre deux régions d'une image. Plus précisément, entre une région sombre et une région claire de cette image. [M-KAD99]. Le réglage du contraste d'une image ne se fait donc plus par incrément sur la palette, mais par un rétrécissement du nombre de valeur de celle-ci. Par exemple si l'on a à disposition une image à 256 niveaux de gris, au lieu d'avoir une palette allant de 0 à 255 par incrément de 1,

0	1	2	3	4	5	6	7	8	9	.....	254	255

Figure 58 : Palette de 256 niveaux de gris

nous aurons une palette allant de 0 à 255 par incrément de 2.

0	0	1	1	2	2	3	3	4	4	.....		255	255

Figure 59 : Palette de 256 niveau de gris avec contraste porté à 2

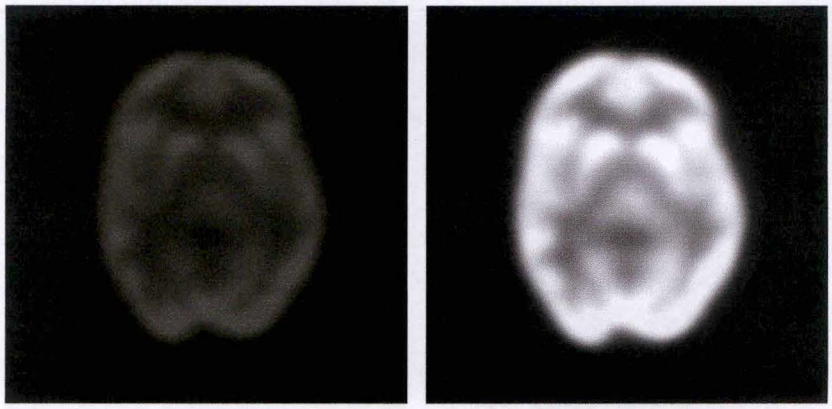


Figure 60 : Image ayant subit une modification de contraste

Point de vue utilisateur:

L'utilisateur a, à sa disposition, la fenêtre de la figure 61, pour régler la luminosité et le contraste de ses images.

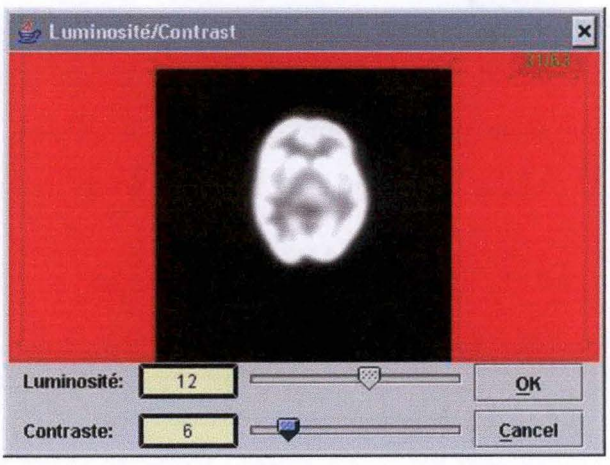


Figure 61 : Fenêtre de réglage de la luminosité et du contraste

Le second outil sert simplement à obtenir le négatif d'une image. Ce qui est utile pour mettre certaines zones plus en évidence.

Principe:

Le principe utilisé est relativement simple. Il consiste à recopier les valeurs de la palette de couleurs utilisée dans le sens inverse.

0	1	2	3	4	5	6	7	8	9	.....	254	255

Figure 62 : Palette de 256 niveaux de gris

→

255	254	253	252	251	250	249	248	247	246	.....	1	0

Figure 63 : Palette de 256 niveaux de gris inversée

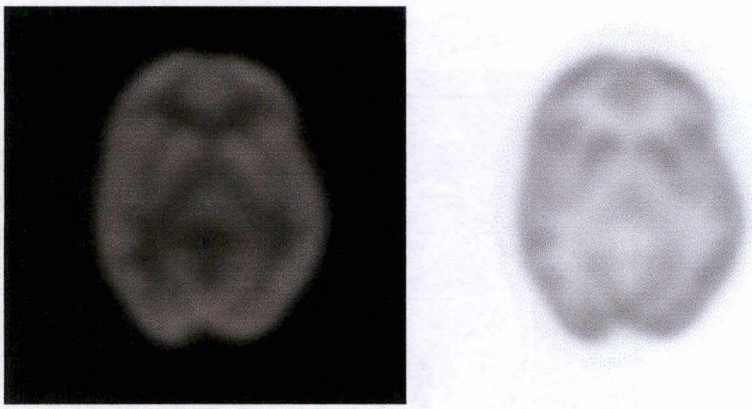


Figure 64 : Négatif d'une image

Point de vue utilisateur:

Il s'agit simplement d'un bouton présent dans le menu de l'application à presser pour inverser la palette de couleurs…

Figure 65 : Menu d'inversion de palette

4.3.4 Présentation des images corégistrées

Outre tous ces outils de visualisation, on observe sur la figure 66, qu'il reste un panel de visualisation qui n'a pas encore été présenté. C'est le panel Corégistration. Une fois la corégistration effectuée, il est possible de visualiser les images alignées de façon particulière. La première, la plus simple, est nommée "parralel display". Elle résulte simplement de l'arrangement de la fenêtre, qui permet d'afficher, en parallèle, les images corégistrées.

Figure 66 : Illustration du parallel display. Les images sont affichées les unes à coté des autres

Ce mode d'affichage est le plus simple que l'on puisse trouver et ne constitue pas, en soi, une révolution. C'est pourquoi d'autres outils précieux sont également présents.

Le premier permet de naviguer dans chaque pile d'images affichées en même temps. Nous l'appellerons "linker".

Principe:

Le principe du linker est de faire en sorte que les piles d'images soient toujours en phase. Grâce à cet outil, la navigation dans les tranches des images se fait en même temps pour chaque pile d'images.

Si le principe est simple à comprendre, il nous a posé un certain nombre de problèmes à l'implémentation. Notamment pour trouver un moyen propre de répercuter l'événement du drag de la souris sur le linker vers les panels de visualisation[16] Néanmoins, l'étude du système événementiel réalisé au point 3.4.2 nous a été d'un grand secours et a permis de trouver une solution élégante à notre problème, en enregistrant les références des panneaux de d'affichage comme listeners de l'objet linker[17].

Point de vue utilisateur:

Pour utiliser le linker, il suffit d'effectuer un drag de la souris sur le composant de la figure 67 au lieu de l'effectuer sur l'une des images.

Figure 67 : Composant "Linker". Ce composant peut être

[16] Ceux présents en figure 43 (Coupe 1, Coupe 2 et Fusion)

[17] Nous invitons le lecteur à aller reconsulter le point 3.4.2 pour lever toute zone d'ombre quant aux termes utilisés.

Enfin, nous clôturons ce tour d'horizon des outils de visualisation en présentant l'ultime outil de visualisation des images corégistrées.

Cet outil permet de superposer deux images suivant différents modes, afin de profiter au maximum des avantages offerts par chacune d'elles.

Le premier mode de fusion implémenté est le mode de fusion par bandes de couleurs.

Principe:

Il consiste à prendre une des composantes primaires[18] de chaque image et de reconstituer une nouvelle image à partir de ces informations. Nous prendrons par exemple, la composante verte de la première image, la composante bleue de la seconde, les fusionnerons afin d'en retrouver la résultante.

La figure 68 illustre un exemple d'image provenant d'une IRM de laquelle on n'a gardé que la composante rouge, fusionnée avec une image du PET-SCAN qui n'a gardé que sa composante verte.

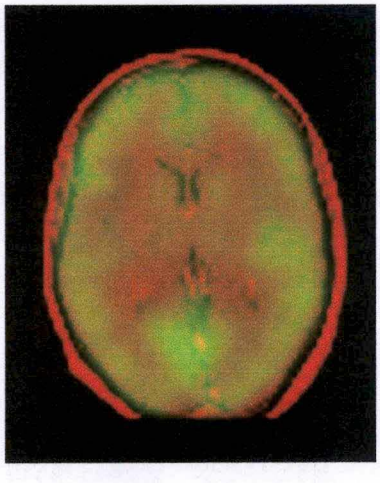


Figure 68 : Image d'IRM fusionnée avec une image du PET-SCAN

[18] Nous avons vu, au point 4.3.1 que les images étaient codées suivant différents modes. Un de ces modes est le mode RGB où une image est codée en utilisant les intensités des couleurs primaires Rouge, Verte et Bleue. D'autres modes existent mais ne seront pas exposés dans ce mémoire. Pour plus d'informations à ce sujet, le lecteur est invité à consulter [M-Van98].

Le deuxième mode de fusion des images corégistrées est le mode de fusion par transparence, qui permet de superposer deux images en attribuant à chacune d'elle des coefficients de transparence α variant entre 0 et 1.

Principe:

Les pixels relatifs à la première image sont multipliés par un paramètre α tandis que les pixels en provenance de la seconde sont multipliés par 1- α. Les valeurs des pixels sont alors additionnés pour donner la nouvelle image.

$$pixNewImg[k] = (pixImg1[k] * \alpha) + (pixImg2[k] * (1-\alpha)), \forall k$$

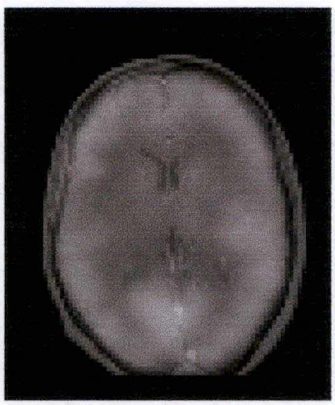


Figure 69 : Exemple de fusion par transparence avec un paramètre alpha de 0,30 pour l'image IRM

Point de vue utilisateur:

La fusion s'effectue aussitôt que les images ont été corégistrées, l'utilisateur n'a alors qu'à choisir le mode de fusion qu'il désire grâce au panneau de la figure 70.

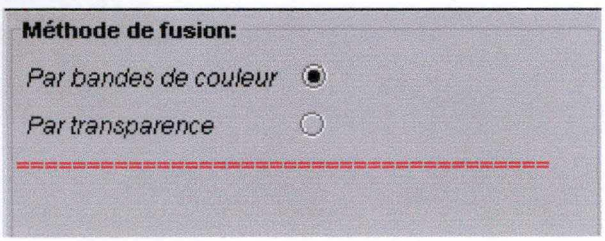


Figure 70 : Panneau de choix du mode de fusion. Ce panneau est visible, dans son contexte, à la figure...

4.3.5 Dernières remarques

Chaque panneau de visualisation est codé indépendamment l'un de l'autre. Ils implémentent simplement une seule et même interface. Seul le panneau Corégistration est immuable car il représente la base de la plate-forme. Sans lui, la plate-forme ne peut fonctionner et n'a d'ailleurs pas de raison d'être. "Visualisation" et "Expert", présentés en 4.3.3 sont des "plugins" gérés par le mécanisme de réflexivité présenté en 3.4.1. Ceci permettra donc, par la suite, à un développeur quelconque n'ayant qu'une connaissance minime du fonctionnement de la plate-forme, de coder ses propres panneaux de visualisation pour, par exemple, ajouter un outil de reconstruction 3D ou un outil de découpe osseuse autour du cerveau…

Figure 71 : Interface principale de l'application

Voici une capture d'écran de l'interface principale de l'application. Les onglets représentent les différents moyens de visualisation disponibles. Les panneaux "Visualisation" et "Mode Expert" sont des panneaux "plugins" ajoutés automatiquement. Leur présence n'est pas obligatoire au fonctionnement de la plate-forme. Ils sont instanciés par réflexivité.

Dernière remarque concernant la plate-forme, le programme a été développé sur base d'une JFrame. Dès lors, rien ne nous empêche de lancer celui ci en applet en étendant la classe JApplet au lieu de JFrame. Il suffit alors de générer un certificat nous donnant les droits d'accès aux disques…

5 Critiques et Discussions

Nous venons donc de faire le tour de l'application réalisée. Nous allons, dans cette partie, tenter de faire la critique des outils implémentés en mettant en avant d'éventuelles faiblesses et, dans la mesure du possible, y apporter des solutions.

5.1 Partie corégistration

L'intégration d'AIR au sein de la plate-forme semble être un succès. JNI a joué parfaitement son rôle et permet l'utilisation du package d'une manière complètement transparente. Les résultats obtenus à l'intérieur de la plate-forme sont semblables à ceux obtenus lorsque les algorithmes sont utilisés en ligne de commande[19]. Il subsiste cependant un problème ennuyant, faisant échouer le processus de corégistration à peu près une fois sur dix. Ce problème est sans doute dû à une fausse manœuvre dans la gestion des accès mémoire que JNI doit effectuer vers JAVA mais n'a pas pu être identifié clairement. Le résultat est un plantage du processus, levant une exception JAVA. L'utilisateur doit alors relancer la corégistration. Ce problème n'est pas encore trop gênant, à ce stade de développement et se révèle être assez rare, mais deviendra inacceptable pour une utilisation de la plate-forme par les utilisateurs finals.

Le deuxième point de discussion porte sur le mode de stockage des données que nous avons revu en 4.1.1. Pour le moment, les images utilisées sont composées de données brutes de type short ou unsigned short. Mais il se pourrait que dans l'avenir, la plate-forme soit utilisée avec des images ayant des données brutes de type float. Cette spécificité a été découverte trop tard et nous laisserons donc le soin au prochain développeur de remédier à ce problème en changeant le tableau d'entiers utilisé pour stocker les données brutes en un tableau de "float", voir de "double", pour prévoir encore de nouveaux types d'images. Le changement ne doit pas poser de problème spécifique.

[19] N'oublions pas qu'AIR s'utilise, par défaut, en ligne de commande, en passant par une console linux ou une fenêtre DOS.

5.2 Partie visualisation

Peu de remarques pour cette partie. Les outils implémentés ne semblent pas pourvus de bug spécifique. Il y aurait néanmoins un point à revoir.

Le réglage de la luminosité et du contraste se fait à partir de la fenêtre suivante,

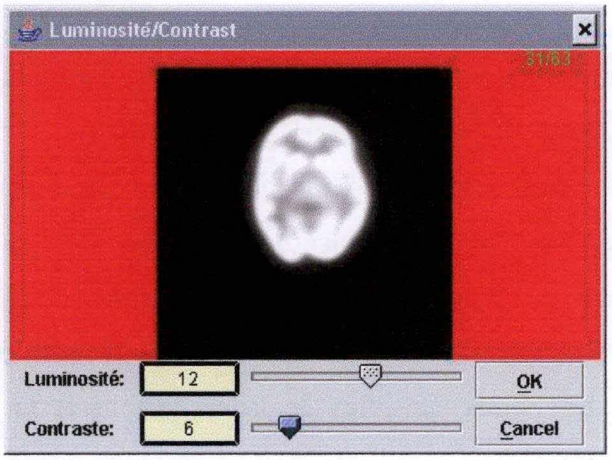


Figure 72 : Réglage de la luminosité et du contraste

Ce n'est peut-être pas le moyen optimal de présenter l'outil et un réglage direct sur les panneaux d'affichage, à l'aide d'un "drag" à la souris serait peut-être souhaitable[20]. Mais ceci n'est qu'une question d'esthétique et de préférence qui n'engendre que des changements minimes et rapides…

[20] Du même genre que pour le zoom, le pan et la rotation… → 4.3.3

Conclusion et vues du futur

Pour conclure ce travail, nous allons résumer les différentes étapes qui ont constitué la rédaction de ce mémoire, nous donnerons ensuite des idées sur les choses importantes qui devront être réalisées par ceux qui reprendront le projet et nous finirons sur une note personnelle.

Dans un premier temps, nous avons mis en place les concepts de base nécessaires à la compréhension de ce qu'est la corégistration. Nous y avons décrit les différents termes utilisés. La partie suivante exposait l'état de l'art en présentant les concepts de la corégistration, et en décrivant l'existant concernant le projet. Les technologies et techniques mises en œuvre ont ensuite été présentées afin de pouvoir introduire notre réalisation et voir comment celles-ci ont pu y être utilisées. Un dernier chapitre vient conclure ce mémoire en donnant une critique des points sensibles de la réalisation et en donnant des pistes de résolution de ces derniers.

Ce programme n'est que le point de départ de la plate-forme de corégistration et beaucoup d'améliorations et d'outils nouveaux pourront y être apportés. Le programme a été écrit surtout dans une optique particulière: être aussi évolutif et générique que possible en mettant en place des mécanisme puissants tels que la réflexivité.

Les buts de celui qui prendra la relève seront multiples. Maintenant que les bases sont posées et que l'on a démontré qu'il était possible d'intégrer un package de corégistration, il faudrait continuer et en intégrer un second, par exemple SPM, qui présente bien des différences de fonctionnement par rapport à AIR mais qui a été conçu en visant les mêmes objectifs.

La partie d'aide à la corégistration présentée en 4.1.2.1 devrait également être développée d'avantage en implémentant des algorithmes de reconnaissance de forme, de détection de contour, de débruitage des images,... Il est fort à parier que JAI prendra, dans cette prochaine réalisation, une place de choix.

Dans la partie visualisation, de nouveaux panneaux de visualisation/travail pourraient également être ajoutés. N'oublions pas qu'ils sont gérés par réflexivité et que l'ajout de tels parties à la plate-forme ne demandera pas au programmeur une quelconque travail de fond sur celle-ci, lui laissant toute possibilité d'élaboration de ses composants.

Un atelier de 'découpe osseuse' plus évolué que le threshold pourrait par exemple être un outil d'une grande utilité. De même, un outil de marquage manuel ou automatique de différentes zones d'intérêt pourrait constituer un outil très intéressant.

N'oublions pas non plus la gestion des données patients, pour le moment laissée de côté mais qui revêtira, dans le futur, une grande importance.

Enfin, une dernière chose extrêmement utile serait l'intégration de ce programme à Telemis®. Pouvoir s'échanger les données et travailler sur les mêmes images apporterait un plus indéniable à ce projet.

Comme nous pouvons le constater, le travail ne manque pas et le chemin est encore long. Mais le projet avance d'année en année et semble prendre forme, confirmant la théorie selon laquelle il serait possible d'élaborer une plate-forme de corégistration assistée…

Sur un plan personnel, ce travail m'a permis de découvrir bon nombre d'outils très puissants dont je n'avais connaissance que par le nom et il m'a donné une nouveau regard sur ma façon de développer et m'a permis d'évoluer dans ma façon de travailler. Je ne saurais qu'encourager les suivants à progresser dans ce projet extrêmement riche et prometteur, tant sur le plan technique qu'humain. Cette réalisation, si elle aboutit, sera en effet un outil d'une grande efficacité pour l'imagerie médicale.

Bibliographie

[M-Van98] Vandermeersch, F. *"Présentation multimodale en imagerie médicale"*, Mémoire de maîtrise en informatique, FUNDP Namur, 1998

[M-Kad99] Kaddour, C. *"Compression des images fixes par fractales basée sur la triangulation de Delannay et la quantification vectorielle"*, Mémoire en informatique, université des sciences et de la technologie Houari Boumediene, Algérie, 1999

[M-Sey02] Seynave, S. *"Corégistration d'images médicales : contexte, modélisation du problème et sa traduction en une première conception orientée objet"*, Mémoire de maîtrise en informatique, FUNDP Namur, 2002

[M-Vae03] Vaesen, O. *"Elaboration d'une plate-forme orientée objet dédiée à la corégistration d'images médicales"*, Mémoire de maîtrise en informatique, FUNDP Namur, 2003

[M-Zuy03] Zuyderhoff, L. *"Composition Automatique de frottis numériques"*, Mémoire de maîtrise en informatique, FUNDP Namur, 2003

[T-Thi97] Thiran, J-P. *"Présentation et recalage d'images tridimensionnelles par squelettes morphologiques"*, Thèse de doctorat en sciences appliquées, UCL Louvain-la-Neuve, 1997

[T-Mäk04] Mäekelä, T. *"Mise en correspondance en imagerie cardiaque multimodale : vers un modèle anatomo-fonctionnel individualisé du cœur"*, Thèse de doctorat en Images et synthèse, Institut National des Sciences Appliquées, Lyon, 2004

[W-CorImg] "La corégistration d'images médicales":
http://www.tele.ucl.ac.be/PEOPLE/OC/coreg

[W-IMED] "IMED – Imagerie médicale":
http://www.ouest-radiologie.com/

[W-ImgVect] "L'image vectorielle":
http://membres.lycos.fr/compressions/vect.html

[W-IntNum] "Introduction à l'image numérique":
http://www.gerbeaud.com/creation/img_num/imagenum.htm
[W-JAI] "What is Java Advanced Imaging?":
http://java.sun.com/products/java-media/jai/whatis.html
[W-JNI] "(JNI) – Java Native Interface":
http://perso.wanadoo.fr/jm.doudoux/java/tutorial/chap029.htm
[W-NecDic] "Comprendre la nécessité de DICOM":
http://dicom.online.fr/fr/dicomwhatfr.htm
[W-NorDic] "La norme DICOM en imagerie médicale":
http://eviewbox.sourceforge.net/JFR98/index.html
[W-RadImg] "Radiographie et imagerie médicale"
http://www.med.univ-angers.fr/discipline/radiologie/Methodes.html

Sites de référence:

Java™ 2 SDK, Standard Edition Documentation
http://java.sun.com/j2se/1.4.2/docs/index.html

Online Java, C#, XML, Python, DSP, and JavaScript Tutorials
http://www.dickbaldwin.com/

Automated Image Registration
http://bishopw.loni.ucla.edu/AIR5/index.html

Java[tm] Advanced Imaging API Documentation
http://java.sun.com/products/java-media/jai/docs/index.html

MRI and PET coregistration A crossvalidation of SPM and AIR
http://citeseer.ist.psu.edu/122048.html

Statistical Parametric Mapping
http://www.fil.ion.bpmf.ac.uk/spm/distrib.html

Annexe I - La réflexivité en JAVA

Voici une partie de l'implémentation de la classe PanelsFactory. Ce code permet de mettre en évidence le fonctionnement du principe de réflexivité tel qu'il a été utilisé dans ce mémoire. Les commentaires présents dans ce code permettent au lecteur de comprendre le fonctionnement de ce mécanisme.

```
public class PanelsFactory extends AbstractButton implements
EventListener,ActionListener,CJPDisplayContainer
{

    /** Creates a new instance of PanelsFactory */
    private String FileName="inidemoi\\panels.ini";
    //Fichier de configuration qui contient les noms des classes que
    //l'on pourra instancier par réflexivité.

    private FileRW fp;

    private Class flt;
    //Objet de type Class.  Il servira à récupérer les classes d'objets à instancier,
    //à partir de leur nom
    private Vector OPanels;
    //Vecteur qui permettra de contenir les panels d'affichage une fois qu'ils
    //auront été instanciés.

    private Constructor cst;
    //Objet de type Constructor.  Il servira à récupérer le constructeur des classes
    //que l'on voudra instancier, au moment de l'exécution.

    private Vector PanelsClass;
    //Vecteur qui contiendra le nom des classes lus du fichier ini
    private Vector PanelsNames;
    //Vecteur qui contiendra les noms à afficher dans l'onglet du JTabbedPane

    private String strtmp;
    private CJPDisplayContainer Parent;

    public PanelsFactory(CJPDisplayContainer parent,JComponent pane,ImageFactory fact)
    {

     Parent=parent;
     PanelsClass=new Vector();
     PanelsNames=new Vector();

     OPanels=new Vector();

     PanelsClass.add("Coreg.Couche4.DisplayPanels.CJPCoreg");
      //Le premier panel est obligatoire, c'est lui qui régit tt le reste...
      //Nous ne le lisons donc pas du fichier de configuration car il doit
```

```
 //obligatoirement être présent ==> il est hardcodé
PanelsNames.add("Corégistration");
 //On ajoute le nom sous lequel ce panneau sera visible dans l'interface

try
{
 fp=new FileRW("Reader",FileName);
 //On ouvre le fichier ini


 try
 {
   while((strtmp=fp.getString())!="eof")
   {
    StringTokenizer str=new StringTokenizer(strtmp,";");
    PanelsNames.add(str.nextToken());
    PanelsClass.add(str.nextToken());
   }

 //On lit les noms des différents panels à instancier depuis le fichier ini
 //Et on les ajoute aux vecteurs

 }
 catch(IOException e)
 {
  String mess="Erreur de lecture du fichier "+FileName+" les panels d'affichage
               auxiliaires n'ont peut-être pas pu être chargés correctement!";

  JOptionPane.showInternalMessageDialog(null,mess,"Alerte",
                                        JOptionPane.WARNING_MESSAGE);
 }
 catch(NoSuchElementException nse)
 {
  String mess="Erreur de lecture des paramètres fichier "+FileName+" les panels
               d'affichage auxiliaires n'ont peut-être pas pu être chargés
               correctement!";
  JOptionPane.showInternalMessageDialog(null,mess,"Alerte",
                                        JOptionPane.WARNING_MESSAGE);
 }

}
catch(IOException e)
{
  String mess="Le fichier "+FileName+" n'a pas été trouvé! Impossible de charger
               les panels d'affichage auxiliaires!";
  JOptionPane.showMessageDialog(null,mess,"Alerte",JOptionPane.WARNING_MESSAGE);
}


//On instancie toutes les classes se trouvant ds le vecteur
//Et on rempli le vecteur devant contenir les instances des panneaux d'affichage
for(int i=0;i<PanelsNames.size();i++)
{
   try
   {
    flt=Class.forName((String)PanelsClass.get(i));
    //On récupère la classe à partir de son nom

    cst=flt.getConstructor(new Class[] {CJPDisplayContainer.class});
    //On récupère le constructeur de la classe récupérée auparavant
```

```
            OPanels.add((CJPPanelsInterface)cst.newInstance(new Object[]{this}));
            //On instancie la classe et on l'ajoute au vecteur contenant les panneaux
            //d'affichage.

            ((JTabbedPane)pane).add((String)PanelsNames.get(i),
                                    (CJPPanelsInterface)OPanels.get(i));

            ((CJPPanelsInterface)OPanels.get(i)).addit(fact);
            //Enfin, on l'ajoute dans l'interface

            this.addActionListener((CJPPanelsInterface)OPanels.get(i));
            //Et on ajoute les listener nécessaire à la gestion des évenements qui
            //pourraient surgir dans ce panneau.
           }
           catch(Exception e)
           {
            System.out.println("Failed to create "+(String)PanelsNames.get(i)+" class");
            e.printStackTrace();
           }
        }
    }

    .
    .
    .
}
```

Annexe II - Illustration du fonctionnement de JNI

Comme nous l'avons mentionné en 3.2.2, l'intégration d'un programme natif à JAVA requiert l'utilisation de Java Native Interface. Nous illustrons ici son implémentation des deux côtés (JAVA et C) en donnant un exemple concret mais simplifié de son utilisation dans la plate-forme. Les parties relatives à JNI ont été mises en évidence et le code a été épuré pour permettre au lecteur de mieux se situer. Mais ce dernier comprendra, sans aucun doute, les difficultés rencontrées lorsque cette étape a été réalisée.

```
public class AirMod
{

   static
   {
     System.loadLibrary("JNInterface");
     //Chargement de la librairie dynamique (dll) JNInterface.dll
     //On le fait en static car, évidemment, on ne doit pas le faire à chaque
     //instanciation de la classe…
   }

   public native int zoom(int args, String[] argv,MedicalImage std);
   //Déclaration de la fonction native zoomer.  C'est à partir de cette déclaration
   //que le fichier d'entête (.h) sera généré pour la partie native du code

   private MedicalImage Std_Img,Rsc_Img,Res_Img;


   public AirMod(MedicalImage std, MedicalImage rsc)
   {
     Std_Img=std;
     Rsc_Img=rsc;
     //Les images médicales
   }


   public int Zoom(String[] args)
   {
      return this.zoom(args.length,args,Rsc_Img);
      //Méthode permettant d'appeler la fonction native à partir de JAVA
   }


}
```

```
JNIEXPORT jint JNICALL Java_Coreg_Couche2_airint_AirMod_zoom(JNIEnv *env, jobject obj,
                                        jint lgth, jobjectArray args, jobject std_img)
{
 char **temp;

 setGlob(env,obj,std_img,NULL);
 temp=exportArgs(args,lgth);

 int val=zoomer_main(lgth,temp);
 freeMem(temp,lgth);

 return val;
}
```

```
char **exportArgs(jobjectArray args,int lgth)
{
 int i;
 char tmp[200];
 jstring jstr;
 char **temps;

 temps=(char**)malloc(sizeof(char*)*lgth);

 for(i=0;i<lgth;i++)
 {
    jstr = (*glob_env)->GetObjectArrayElement(glob_env, args, i);

    strcpy(tmp,(*glob_env)->GetStringUTFChars(glob_env, jstr, 0));
    temps[i]=(char*)malloc(strlen(tmp));
    strcpy(temps[i],tmp);
    printf("\n%s",temps[i]);
 }

  (*glob_env)->ReleaseStringUTFChars(glob_env, jstr, tmp);

  return temps;
}
```

```
__declspec(dllexport) int zoomer_main(int argc, char *argv[])

{
            .
            .
            .
            .

                  AIR_Error errcode=AIR_do_zoomer(argv[0], argv[1], argv[2], air_file,
                                                  imageout, ow, airow);
                  if(errcode!=0){
                        AIR_report_error(errcode);
                        exit(EXIT_FAILURE);
                  }

      return EXIT_SUCCESS;
}
```

```
AIR_Error AIR_do_zoomer(const char *program, const char *input, const char *output,
const char *air_file, const AIR_Boolean imageout, const AIR_Boolean ow, const
AIR_Boolean airow)

{
            .
            .
            .

      if(imageout){
            AIR_Pixels  ***datain=NULL;
            AIR_Pixels  ***dataout=NULL;


            AIR_Error errcode;

            datain=JAVA_load(input, &air1.r, 0, &errcode);
            if (!datain){
                  free_function(datain,dataout);
                  return(errcode);
            }

            .
            .
            .

                  AIR_Error errcode=JAVA_save(dataout,&air1.s,output,ow,input);

                  if(errcode!=0){
                        free_function(datain,dataout);
                        return(errcode);
                  }

            free_function(datain,dataout);
    }

    .
    .
    .

      return 0;
}
```

```
AIR_Pixels ***JAVA_load(const char *infile, struct AIR_Key_info *stats, const
AIR_Boolean binaryok, AIR_Error *errcode)
{
            .
            .
            .
           globalscale=JAVA_open_header(infile,&fps,stats,flags);
           if(fps.errcode!=0)
           {
            printf("KC");
            *errcode=fps.errcode;
            return NULL;
           }
            .
            .
            .
```

```
        //on regarde à quel fichier on a affaire et on charge l'objet en
        //conséquence

        jobject img;
        if(infile[0]=='s')
         img=glob_std_img;
        else
         img=glob_rsc_img;

         //on récupère la classe de l'objet de type MedicalImage
            jclass cls = (*glob_env)->GetObjectClass(glob_env, img);

         //et on lit tous les champs de l'objets qui nous intéresse ouf! :-)

            jfieldID fid1 = (*glob_env)->GetFieldID(glob_env, cls, "pixbuff",
                                                    "[I");

            jobject jtab = (*glob_env)->GetObjectField(glob_env, img, fid1);


        jint *body = (jint *)(*glob_env)->GetPrimitiveArrayCritical(glob_env,
                                                                    jtab,NULL);

        int i,j,k;
        int tempz,tempy;

        .
        .//On effectue des opérations sur Jtab…
        .

        (*glob_env)->ReleasePrimitiveArrayCritical(glob_env, jtab,
                                                   body,JNI_ABORT);

        return image;
}
```

```
float JAVA_open_header(const char *infile, struct AIR_Fptrs *fp, struct AIR_Key_info
                       *stats, int *flag)
{
        /*Récupérer tt ça de java*/
        /*-----------------------*/
        double value=1.0;
        int glmin,glmax;
        jobject img;
        fp->errcode=0;
        //std si le fichier commence par s(rajouter ce prefixe au nom du fic en
        //java) ou rsc c'est selon...
        //voir qd meme si c en relatif ou en absolu...

        //on regarde à quel fichier on a affaire et on charge l'objet en
        //conséquence

        if(infile[0]=='s')
         img=glob_std_img;
        else
         img=glob_rsc_img;
```

```
        //on récupère la classe de l'objet de type MedicalImage

        jclass cls = (*glob_env)->GetObjectClass(glob_env, img);
        //On prépare l'appel a la méthode qui permettra de retrouver les infos
        //du header
        jmethodID m_id = (*glob_env)->GetMethodID(glob_env, cls,
                          "getMedicalImageInfos",
                          "()LCoreg/Couche1/MedicalImageInfos;");

        if (m_id == 0)
        {
          printf("bouh ben ça a pas marché ;-(");
          return 1;
        }

        //on appel la méthode et on récupère l'objet de type MedicalImageInfos
        jobject img_inf = (*glob_env)->CallObjectMethod(glob_env, img, m_id);

        //on récupère la classe de cet objet
        jclass infcls = (*glob_env)->GetObjectClass(glob_env, img_inf);


        //et on lit tous les champs de l'objets qui nous intéresse ouf! :-)
         jfieldID fid1 = (*glob_env)->GetFieldID(glob_env, infcls, "bits", "S");
        (jint)stats->bits = (*glob_env)->GetShortField(glob_env, img_inf, fid1);


        fid1 = (*glob_env)->GetFieldID(glob_env, infcls, "x_dim", "S");
        (jint)stats->x_dim = (*glob_env)->GetShortField(glob_env, img_inf, fid1);

        fid1 = (*glob_env)->GetFieldID(glob_env, infcls, "y_dim", "S");
        (jint)stats->y_dim = (*glob_env)->GetShortField(glob_env, img_inf, fid1);

        fid1 = (*glob_env)->GetFieldID(glob_env, infcls, "z_dim", "S");
        (jint)stats->z_dim = (*glob_env)->GetShortField(glob_env, img_inf, fid1);

        fid1 = (*glob_env)->GetFieldID(glob_env, infcls, "x_size", "D");
        (jdouble)stats->x_size = (*glob_env)->GetDoubleField(glob_env, img_inf,
                                                             fid1);
        fid1 = (*glob_env)->GetFieldID(glob_env, infcls, "y_size", "D");
        (jdouble)stats->y_size = (*glob_env)->GetDoubleField(glob_env, img_inf,
                                                              fid1);
        fid1 = (*glob_env)->GetFieldID(glob_env, infcls, "z_size", "D");
        (jdouble)stats->z_size = (*glob_env)->GetDoubleField(glob_env, img_inf,
                                                              fid1);

         fid1 = (*glob_env)->GetFieldID(glob_env, infcls, "min", "I");
         (jint)flag[0]=(jint)glmin=(*glob_env)->GetIntField(glob_env, img_inf,
                                                             fid1);
         fid1 = (*glob_env)->GetFieldID(glob_env, infcls, "max", "I");
         (jint)flag[1]=(jint)glmax=(*glob_env)->GetIntField(glob_env, img_inf,
                                                             fid1);

            .
            .

        return value;
}
```